The development of a colour graphics workstation

Chandler, Simon Robert

Award date:
1987

Awarding institution:
University of Bath

THE DEVELOPMENT OF A COLOUR GRAPHICS WORKSTATION

Submitted by Simon Robert Chandler B.Sc.(Hons.)

for the degree of Ph.D.

of the University of Bath

1987

COPYRIGHT

Bath, March 1987.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

This dissertation describes the development of a general purpose, single user workstation for use in computer aided design applications. The aim throughout the project has been to produce an open system with standard interfaces that affords high portability of graphics applications programs. The bulk of the work has involved the development of high resolution colour graphics systems and their supporting system software. These systems have then been integrated with existing 16/32 bit multi-processor computer systems to achieve the final design goal. Testing of the workstation for standards conformance at the interface level was achieved by porting a large molecular modelling package from a mainframe computer/graphics terminal environment onto the workstation.

The virtual device interface for the graphics systems was chosen to be the I.S.O. and A.N.S.I. adopted graphics standard called the Graphical Kernel System. This applications interface has been implemented, firstly for the TRIPOS operating system, and latterly for the UNIX operating system. These two different operating systems were evaluated in terms of facilities supported and their acceptance in the commercial marketplace, both of which would have directly affected program and programmer portability. UNIX was finally adopted as the operating system for the workstation.

Leading edge technology has been used to implement the graphics systems, both of which are based on specialised integrated circuits designed for this application. At the lowest level, driver software has been written to integrate these graphics devices with their operating systems so that they may be interfaced to the functionality of the higher level Graphical Kernel System. Implementing the Graphical Kernel System has involved the development of a

large suite of programs, many of them involving complex algorithms concerning the generation of drawing primitives and of picture manipulation.

The validity of choosing the Graphical Kernel System as the applications interface for portability was confirmed by demonstrating the molecular modelling program running on the workstation. This demonstration also proved that today's microprocessor based workstations are capable of achieving the per-user performance of mainframe machines.

Acknowledgements

The work presented in this thesis was carried out under the supervision of Mr. A.R. Daniels, Senior Lecturer in the School of Electrical Engineering, University of Bath, England. The author wishes to express his gratitude to Mr. Daniels for his constant interest, enthusiasm and guidance.

The author would like to thank all the members of staff and collegues of the School of Electrical Engineering, University of Bath for their guidance and help. Financial support from the Science and Engineering Research Council is gratefully acknowledged.

The author is indebted to Dr. Murray-Rust of the Glaxo Research Centre, Greenford, for supplying the GLXMOD suite of molecular modelling programs. Gratitude is also extended to the Rutherford-Appleton Laboratories, Abingdon, for supplying their implementation of the Graphical Kernel System and also to the British Technology Group for their financial backing of this project.

LIST OF CONTENTS

CHAPTER 1

SYSTEM REQUIREMENTS FOR INTERACTIVE WORKSTATION DESIGN

1.1 The Development of Workstations for Computer Aided Design.

The history of electronic computing only spans some 40 years, but in that time the advances in technology have accelerated the deployment of computers to the extent where most of us daily come into contact with them, either at first hand, or within some embedded application. Interactive computing applications, particularly in the field of Computer Aided Design(CAD) also require a powerful Man Machine Interface(MMI). The communication between computer and operator is enhanced by the effective use of graphical display devices. Thus, the development of these graphics systems and their supporting software is crucial in attaining effective interactive computing. Single user, desk top computers, with the ability to support powerful CAD applications have recently emerged. These machines have become known as "Workstations".

This chapter investigates the advances that have been made in interactive computing which have brought about the development of the computer workstation. Many of the first commercial workstations to appear were turnkey systems and were therefore inflexible and could not be used to replace mainframe based CAD systems. Hence, the features that are required to achieve an open workstation system that is truly general purpose have been identified so that these may be included in the subsequent workstation design.

1.2 Computing Systems.

The computer systems designer has received considerable support over recent

years from Integrated Circuit(IC) manufacturers who have been designing and supplying devices that not only contain basic logic gates but use a much higher level of integration, in order to provide functional blocks such as Arithmetic Logic Units(ALU) and shift registers. Today, Very Large Scale Integration (VLSI) devices have in excess of 100,000 transistors on a single substrate yet their power dissipation is only of the order of a watt. The reduction in trace and feature size of IC's has reduced their internal capacitive and inductive effects thus enabling higher clocking frequencies to be achieved. Many of today's VLSI devices can be clocked at 20MHz or more. All these factors have helped bring about the very powerful, yet necessarily small devices that are required when designing single user computer workstations.

Integrated circuit manufacturers produce devices that allow computer design to be achieved, both at a sub system level, using what are known as bit-slice devices[1], and, at the system level using microprocessor devices.

Computer system design at the sub system level is used when the machine architecture or instruction set must be optimised for a particular application. It allows the configuration of data bus and address bus size through the selection of functional blocks such as 4 bit cascadable ALUs and microprogram controllers. This approach also allows optimisation of the machine instruction set since the microprogram control store must be programmed by the system designer. Design at the sub system level is still popular in the specialised area of computer graphics where the hardware architecture and instruction set may be tuned for high performance. This requirement in graphics applications is reducing as IC manufacturers produce specialised VLSI devices for this application.

For single user desk top computers, design at the system level is essential

since it reduces the system size, power consumption and design time. The main functional component of this design is the microprocessor. This device generates all the control, address and data signals required to interface to external memory and input/output devices. Microprocessors have a fixed instruction set with either hard wired logic for instruction decoding or a micro-programmable control store that has been programmed by the IC manufacturer. To complement the range of microprocessors, a manufacturer will provide a set of compatible support devices for functions such as direct memory access, memory management and serial/parallel input and output. Microprocessors have grown from simple 4 bit machines to 32 bit devices employing micro-coded techniques to control internal functions and support comprehensive instruction sets that are well suited to supporting today's high level programming languages. Microprocessor systems now provide the power of a mini computer but at a fraction of the cost and size. The Motorola MC68020[2] for example has 32 bit data and address busses, both internally and externally. It also supports a co-processor interface, and, currently the MC68881 floating point processor[3] may be used to improve floating point intensive tasks.

IC manufacturers have also expended considerable effort into producing the memory devices for computing systems. The availability of high density, cheap memories has also been instrumental in the emergence of powerful interactive computers. There are two main ways in which data storage can be effected in semiconductor memories, with each method yielding a different set of device characteristics. The first method combines two transistors to form a bistable memory which has fast access times and a simple interface. Devices using this construction are known as static memories and can currently be obtained with packing densities to 32k x 8 bits with access times as low as 25ns. The second method provides a much higher packing density per storage bit by using the

charge held on a capacitor to indicate a zero or one state. The drawbacks of this device are slower access times and increased overheads brought about by the requirement to refresh the charge on the memory cells at regular intervals to avoid data loss. These devices are known as Dynamic Random Access Memories(DRAM) and current packing densities are 1 Mbits per device with cycle access times of approximately 250ns.

The trends of reducing feature size to provide more functionality per device will continue into the future as Ultra Large Scale Integration (ULSI) devices emerge. Clock speeds will increase, but power consumption will decrease and desk top and hand held computers will become as powerful as the mini and mainframe machines of today. New machine architectures will support parallel processing to improve performance further.

1.3 Operating Systems.

The operating system of a computer comprises a set of programs to control the access to, and usage of the machine's resources. Early operating systems were only capable of running a single user task at a time, and hence users requiring computer time had to queue their jobs to run in batch mode. To achieve interactive use, an operator would therefore need be the only user of the machine. Since early computers were very expensive, their use in interactive applications was not economical and any CAD was limited to the design checking phase.

Multi-user operating systems have enabled the resources of a mainframe machine to be shared amongst many simultaneous users. The response time to a request from an operator is dependent upon the processor load, therefore under worst case load conditions the machine can become so slow as to severely

degrade operator efficiency. Multi-user operating systems are, however significant in the appearance of economical interactive computing.

The microprocessor has enabled single user machines to be economically built, and many single task operating systems have been designed to run on these machines. Microprocessors are now powerful enough to support multiple tasking, multiple user operating systems with similar facilities to those of mainframe machines. A workstation for CAD should support a multi-tasking environment so that the system may efficiently handle the many devices that will be simultaneously serviced.

The user interface of operating systems has been improved through the use of commands that are close to their English equivalent and through the provision of help responses when the operator is having difficulty. Today, menu driven windowing systems are becoming popular. The operator interacts with the computer using an input device, such as a mouse, so that the conventional keyboard becomes almost redundant. For graphical interaction, a workstation should support several input devices so that the most natural device may be chosen for a particular style of input. The response time of the computer to a request from the operator should be as fast as possible. A large proportion of interactive computer users only deal with alphanumeric oriented displays, however, CAD is a growing application area, and graphical interaction must offer the same response times as these character based systems.

If a workstation is to support the development of CAD applications it must additionally provide the systems programmer with a suitable scientific high level programming language and programming development tools. The choice and use of this language will affect the portability of CAD programs developed on the workstation.

1.4 Interactive Graphics Display Devices.

As computers and their operating systems have advanced, so have developments in the field of computer graphics been made. This parallel effort may be logically split between that made in developing graphics hardware devices and that expended in the development of software systems to support the hardware. Since the appearance of microprocessors and, indeed, multi-user operating systems is a fairly recent event, the impetus for graphics hardware development has only been strong since the late fifties, or early sixtes when work on interactive graphics applications really began. The Sketchpad project[4], which began in 1961 is one of the first applications of interactive graphics. The equipment used to support the Sketchpad project seems very crude by today's standards. The evolution of computer graphics equipment since that time illustrates how and where performance has been improved. These trends are important factors influencing the appearance of single user workstations.

To categorise computer display hardware it is useful to introduce the concept of interactive and non interactive devices. An interactive device will permit almost instantaneous modification of its display surface whilst a non-interactive device will only be suited to obtaining hard copies. Many new display devices for interactive use are beginning to appear in commercial and military equipment. Liquid Crystal Displays(LCD) and plasma panels are two examples of devices that have been developed to replace the conventional Cathode Ray Tube(CRT) which is bulky, power hungry and fragile. Nevertheless, the CRT remains as one of the few thermionic valves that is still widely used today. A CRT produces an image on a phosphor coated screen by using electrons to bombard the phosphor which then emits light. Colour may be

generated by coating the CRT display surface with triads of red, green and blue light emitting phosphor dots. Three electron guns and a shadow mask are used to ensure that the electrons from a particular gun will only strike the phosphor dots of one colour. Colour tubes are more expensive and bulky than monochrome tubes and require three drive circuits for the red, green and blue guns. The fabrication problems of producing very fine shadow masks with sufficient strength and the ability to withstand temperature rises caused by electron bombardment have prevented colour tubes reaching the resolution of monochrome tubes.

As the electrons travel along the CRT, they are deflected, either by electrostatic or electromagnetic means, such that every point of the display surface may be bombarded. The two scanning methods used to generate images on a CRT screen are known as vector scanning and raster scanning respectively. These two techniques are fundamentally different and yield graphics systems that have different characteristics.

Vector scanning, or random scanning as it is also known, was the first method employed for interactive graphics use. Images were produced using electrostatic deflection by the application of voltages to X and Y deflection plates within the CRT. Because of this, vector scanning systems were most suited to applications requiring images to be constructed from a series of lines, which resulted in the use of wire frame models to represent three dimensional objects. The complete image must be continually scanned at a repetition rate which ensures that picture flicker does not occur. The earliest vector scanning systems[5] required the host computer to calculate all the values to be fed to the D/A convertors and store these internally so that they could be continually read out to the display device. The problems with this approach were excessive loading of the

computer and the requirement for expensive memory devices to store the values for the D/A convertors.

Cheaper graphics systems became available with the introduction in 1968 of the storage tube display[6]. This device was again random scanning, but it did not require continual refreshing, hence, the very expensive picture store was not required. Unfortunately, the storage tube had the disadvantage that selective erasure of parts of the picture could not be achieved.

Most of the interactive display systems in use today employ raster scanning techniques to generate images on a CRT. This is the same method as used in commercial television equipment and hence, considerable research effort has been expended by manufacturers to reduce costs and improve picture quality. Electromagnetic deflection is used to produce a picture by repeatedly scanning the electron beam from the left hand side to the right hand side of the CRT, while sequentially moving line by line down the screen. In this way, the whole display surface is addressed, each complete scanning sequence being known as a frame. The frame update rate must be sufficient to avoid the screen from flickering. The frame frequency for commercial television equipment is 50Hz[7] and a similar frequency is usually adopted for high resolution graphics systems. As the frame is being scanned, the intensity of the electron beam is modulated by a video signal, the magnitude of which corresponds to the intensity of the displayed picture. Flyback must occur when the electron beam reaches either the right hand side, or the bottom of the screen, and during these retrace periods, the video signal must be blanked. To maintain synchronization, additional pulses are fed to the display monitor to temporally indicate the start of each line and frame. These signals are sometimes combined to produce a composite video signal. The scanning sequence, and corresponding composite

video signal for a raster scan system is shown in figure 1.1. A technique known a interlaced scanning must sometimes be used when the bandwidth of the video signal for normal scanning would be too great. Each frame is split into two parts known as the odd and even fields which contain all the odd and even lines of the frame respectively. Each field is displayed at the repetition rate that would normally be required for a complete frame, but because the two fields are interlaced the picture flicker is not noticible. Hence, interlacing enables the video bandwidth to be halved without introducing flicker. Problems can however occur for computer generated images, since adjacent lines in odd and even fields may be assigned widely varying intensities. A long persistence phosphor should therefore be specified for the CRT when using interlacing on computer graphics equipment.

For a computer to be able to generate a picture on a raster scanning display device, it must store a representation of the picture which has been quantized, firstly into lines which will map onto the display scan lines, and secondly into elements along each scan line. These picture elements will be used to generate the video signals that drive the display monitor, and ultimately the intensity of the corresponding area of the displayed image. Each video frame is therefore represented as an X-Y matrix of picture elements, or pixels. Each pixel must therefore be stored by the computer in some way. The per-pixel storage requirement will depend upon its representation, for example, a black and white image consisting of pixels whose intensity is either black or peak white will require only a single data bit per pixel, whilst an image that has been quantized into 256 intensity levels will require 8 data bits per pixel. Graphics systems that store the representation of an image in this way are often known as bit mapped systems. To achieve a reasonable picture resolution, large amounts of computer memory will be required, and hence bit mapped systems

have only been a viable solution since the appearance of high density low cost DRAM memory devices. As these devices have increased in density, so have the number of pixels representing the display of raster scan systems. Early systems would typically have a resolution of up to 512x512 pixels, with a scanning sequence that conformed to commercial television standards. These systems are classed as low resolution, whereas, increasing the scanning frequency yields medium resolution systems at approximately 750x750 resolution and high resolution systems at above 1000x1000 pixels.

A low cost general purpose workstation should use a bit mapped raster scanning display system in order to keep costs to a minimum. There is a wide variety of commercially available raster scanning display monitors, and VLSI devices specifically for bit mapped systems. A high resolution system should be used to prevent quantization effects from imparing the quality of the image. A colour system should be chosen, with the number of intensity levels for the red, green and blue video signals being optimised for cost and performance.

1.5 Programming Graphics Systems.

Just as operating systems have been developed to support a particular manufacturers computer equipment, so have graphics software systems been written to support graphics hardware through the implementation of applications interfaces for popular programming languages. A common set of techniques and algorithms have evolved for these systems, even though the equipment they address have had differing characteristics. The following section investigates some of the fundamental techniques and algorithms that are applicable for graphics systems. Techniques for computer graphics have evolved as hardware has become available and designers have attempted to provide facilities to improve realism, increase speed or improve the performance

of an interactive system.

The problems relating to picture specification and viewing have been tackled mainly in terms specific to vector devices since these were the first to be widely used. Many of the techniques developed with vector devices have been retained for raster scan bit mapped systems but now additional features are needed to cope with filled areas and multiple colour shading.

In general, computer generated images will be produced from an application program whose programming language provides graphical facilities, either as an extension to the language syntax, or as a subroutine library to which the application program may be linked. Many such graphics systems have been produced, and some of these are introduced below. There are many similarities between these systems, and in practice, the implementation of a general purpose graphics system will require certain fundamental techniques and algorithms to be followed. This section studies some of the techniques and algorithms that are applicable to two dimensional viewing, and, in many cases these may be extended to permit three dimensional viewing.

1.5.1 Graphics Primitives.

Graphics primitives are the fundamental constructional components from which an application may build any image. Primitives are chosen to reflect the most commonly required graphical operations. For example, line drawing and text generation. In some cases, graphics hardware may not be capable of directly generating some of the general purpose primitives that are required by a graphics system and the responsibility falls to the host processor which must run emulation software to map the higher level general purpose primitives onto the low level functions of the hardware. As an example, a raster scan system

may only be capable of plotting a single point at a specified x,y position, hence, the constituent pixels of the primitives must be calculated in some way. For bit mapped systems this method of drawing primitives is known as scan convertion.

1.5.2 The Viewing Pipeline.

The viewing pipeline of a graphics system takes the user's application representation of an image as input, and transforms this representation into a form that can be accepted by the display device. Figure 1.2 shows the stages in a typical viewing pipeline. The pipeline contains a set of transformations, rotations, clips and perhaps perspective transforms in three dimensional systems to map the users world coordinate address space onto that of the two dimensional screen coordinate address space. There may be intermediate coordinate address spaces at which storage is performed. The viewing pipeline must be traversed by every coordinate of a primitive and hence requires considerable processing time particularly since the intermediate address spaces may be held using floating point values. For interactive applications the processing associated with the viewing pipeline must be performed quickly enough to produce the effect of instantanious screen update. Some equipment manufactureers have employed specialized hardware to implement the viewing pipeline[8].

1.5.3 Windowing.

The windowing transformation for 2D images is illustrated in figure 1.3. It allows the user the facility to map an area of his world coordinate space onto any area of the display surface. A mapping of this type is defined by,

equation 1 $$x_s = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} \left[x_w - W_{xl} \right] + V_{xl}$$

equation 2 $$y_s = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} \left[y_w - W_{yb} \right] + V_{yb}$$

where the terms used are as follows,

x_s and y_s	target screen coordinates.
V_{xr}, V_{yt}, V_{xl}, V_{yb}	diagonal vertices of viewport.
W_{xr}, W_{yt}, W_{xl}, W_{yb}	diagonal vertices of the world coordinate space to be mapped onto the viewport.

Calculation of screen coordinates from the above equations may be achieved more quickly by rewriting (1) and (2) as, respectively,

$$x_s = A_{xw} + B \quad \text{and} \quad y_s = C_{yw} + D$$

The variables A,B,C and D are constant for a particular transformation and hence need only be calculated once when the transformation is changed.

1.5.4 Picture Manipulation.

Since graphical objects are often represented using many primitives, it is very convenient to have a mechanism whereby a group of primitives may be placed in a named structure so that the structure may be operated on as a whole. This mechanism is often known as picture segmentation, or sub-picture manipulation. Typical operations that may be performed on these sub-picture structures include transformations, hiding, revealing and deletion. For the purpose of interaction, one of these structures may be identified on input by the operator pointing to any one of its constituent primitives.

1.5.5 Graphical Input.

For graphical systems, the usual alphanumeric keyboard is inflexible for efficient user interaction. Instead, the operator must be capable of inputting objects that are semantically linked with the problem he is dealing with. Typically, these are two or three dimensional coordinates that represent a point in his world coordinate space. Physical input devices with two or three axes of movement are used to position a cursor on the display device so that its position may be identified with the objects that are displayed there.

1.6 Algorithms for Computer Graphics.

Since the speed of response is of prime importance in interactive graphics applications, much research effort has been expended in designing efficient algorithms for the graphics system. The following sections will discuss those algorithms that have been implemented in this study. Some of these algorithms have also been implemented by IC manufacturers on specialised devices, and where possible these devices have been exploited to minimise the processing load on the workstation microprocessor.

1.6.1 Algorithms for Primitive Clipping.

If a primitive extends outside of a viewport it must be clipped at the viewport boundary to prevent it interfering with information in adjacent viewports. Algorithms have been designed to efficiently solve the problem of identifying the clipped coordinates of primitives that stray beyond the clipping rectangle. A line clipping algorithm by D. Cohen and I. Sutherland is described in [13], which capitalises on the fact that clipping to a rectangle only ever generates one visible line segment.

The display space is firstly partitioned into nine areas by the clipping boundaries, which, in this case are the edges of the viewport. Each area is given a unique identifier called an outcode to enable its position to be noted. Figure 1.4 shows the partitioned display space together with the identifiers for each area. These identifiers are in a binary format since each of the constituent bits are used to represent which side of the clipping boudaries they lie on. For example, if an area is above the top clipping boundary it will have bit 3 set, and if it lies to the left of the left clipping boundary it will have bit 0 set. Simple logical operators may therefore be used to assertain which area is being tested by examining which bits of the area identifier are set. The area in the centre of figure 1.4 has a bit pattern of all zeros and represents the viewport, which is the only area where primitives will be visible.

The vertices of a vector to be clipped are tested to establish which of the nine clipping regions they lie in. If they both have an identifier of zero then they must both be visible and the vector may be drawn without clipping. If the logical AND of the two identifiers does not give a zero result then the vector must lie completely outside the clipping boundary and may be trivially rejected. If the logical AND is zero, then one of the vertices may be inside the clipping boundary, or the vector may cut across the viewport, but have two invisible vertices. In either of these cases the vector cannot be trivially accepted or rejected and must be clipped to the viewport. Clipping is performed against one boundary at a time, and the testing phase re-entered. This test-then-clip process is continued until both vertices of the vector have an area identifier of zero, at which point the vector may be drawn.

Polygon clipping is an extension of the line clipping algorithm described above. Problems arise if the polygon is convex, as shown in figure 1.5 because it

may fragment, with each new polygon being joined by degenerate edges. The algorithm by Sutherland and Hodgeman[10] is capable of dealing with convex polygons. It is reentrant, and deals with the vertices of the polygon rather than its edges. The flow chart for the first phase of the algorithm is shown in figure 1.6. Except for the first vertex which has no predecessor, the algorithm moves around the figure using the current and previous vertices to decide if the current vertex may be trivialy accepted or rejected, or, if clipping must occur. Hence, a vertex may generate zero, one or two new vertices. Two registers, named S and F in figure 1.6 are required by the algorithm. A flag is also required to indicate whether the current vertex is the first vertex of the figure. When all the vertices have been processed by phase one, the second phase is entered which will ensure that the clipped figure is closed correctly. The flow chart for this phase is shown in figure 1.7.

The algorithm could clip all the vertices of a figure against a single edge and store all the new vertices ready for clipping against the following edges, but this would require unspecified amounts of storage between each clipping stage. Since the vertices are always ordered correctly, a vertex that is found to be on the visible side of one clipping edge may immediately be passed on to the subsequent clipping stages. Since the algorithm shown in figures 1.6 and 1.7 is recursive, it may be re-entered when a vertex is found to be visible, or if a new intersection is calculated by the previous stage. The two stages "output I" and "output S" of the algorithm are reentrant calls, with the total number of nested levels reflecting the number of edges to be clipped against. Each clipping edge is associated with a particular level and has its own registers for S, F and the first flag. Hence, vertices pass from level to level until they are either discarded, or, if they reach the lowest level they are output as a valid point comprising the clipped primitive.

1.6.2 Algorithms for Scan Conversion.

The topic of scan conversion is concerned with the generation or simulation of primitives on a discrete surface by the calculation of the parts containing each primitive. For example, to display a vector on a bit mapped display will require all the pixels lying under the vector to be calculated and modified. For example, a simple algorithm based on taking the floating point representations of dy and dx of the vector, and adding these successively to its starting point will yield the required pixel coordinates comprising the vector. This method is satisfactory, but slow due to the floating point manipulations required. The discrete nature of the display device should enable an algorithm based entirely on integer arithmetic to be devised. One such algorithm, by Bresenham[11], has been widely implemented for high performance systems using dedicated hardware. Digital Differential Analysers(DDA)[12] are also used to scan convert primitives. They are capable of drawing both straight and curved lines and function by operating upon the differential equations of the primitives.

Vector systems and pen plotters are constrained to display solid areas by hatching but raster scan bit mapped systems permit solid areas to be scan converted to completely fill the required area. Two different methods of scan converting solid areas have been developed. The first method is called boundary filling and involves scan converting the boundary by one of the vector algorithms. A point within the boundary must then be identified, and used as the seed point from which filling may commence. The data held within the frame store is read, and compared with that of the boundary. If the boundary was not found, then the point is set with the required fill data, otherwise, the pixel point is incremented to the next scan line and the algorithm proceeds. The boundary fill algorithm may be implemented in hardware, but it does require

some external processing to calculate the initial seed position. The boundary fill is not a general purpose area fill, since it will not resolve self intersecting polygons unless multiple boundary fills are made using the inside points of each of the degenerate polygons.

A second method, not readily implemented with simple hardware is capable of handling self intersecting polygons in a single operation[13]. The area to be filled is split into scan lines, and all the pixels along the line within the polygon are calculated simultaneously. It is efficient because it takes advantage of the coherence of pixels along a scan line. The algorithm may be implemented entirely using integer arithmetic, but it can require large amounts of temporary storage if complex polygons, or high resolution displays are used.

Consider area filling the polygon shown in figure 1.8. The first phase of the algorithm involves the preparation of a Y bucket which contains a list of vectors comprising the polygon, ordered by their maximum Y coordinate value. Figure 1.9 shows the Y bucket and its associated data structures built from the polygon shown in figure 1.8. When the Y bucket has been initialized, the scan conversion can commence, starting from the maximum scan line and decrementing by one on each iteration of the algorithm until the bottom scan line is reached. For each iteration, the Y bucket is accessed and, if a vector has an end point on the scan line it is transferred from the Y bucket to an active edge list. The vectors contained in the active edge list are used with a vector scan conversion algorithm to calculate their constituent pixels for each consecutive scan line. The edges, so calculated in the active edge list are ordered by their x values after each iteration so that crossing vectors are handled properly. When re-ordering has been done, the active edge list is scanned from minimum to maximum x valued edge, and every other line between these edges

filled in. When the minimum scan line for a vector is reached, it is removed from the active edge list. Figure 1.10 illustrates how the active edge list is used to fill the appropriate pixels along a scan line.

1.7 Graphics Systems.

This section investigates the processing hardware of graphics systems. It shows how these systems have developed and where optimizations in their performance have been made. The different requirements for vector and bit mapped raster scan systems are examined to indicate which methods should be employed for CAD workstation systems.

1.7.1 Vector Scanning Systems.

The Sketchpad project mentioned above used an electrostatically deflected CRT or Scope as they were also known that had a ten bit per axis resolution and a maximum spot display speed of about 100000 per second. All the spot locations were stored in the computer memory, each spot being displayed by a single output instruction. The computer was responsible for the calculation of all the spot locations on the display screen as well as the interaction and display list management and the handling of other input/output devices. Sutherland estimated that approximately 75% of the total processing time of the Sketchpad system was taken by the calculation of the dot parts of the picture composition components. Sutherland proposed the use of a separate incremental curve drawing processor that required only the minimum of information about a picture primitive. For example, only the endpoints of a line would need to be stored by the host computer. This would offload processing from the host computer and reduce the number of input/output instructions pertaining to the graphical communication.

The vector scanning systems available today have developed in three stages according to Sutherland's suggestion. Each stage may be identified by the functionality that was available from its display processors. Initially, devices took their instructions and data from the host processor's memory area and used these to generate the x and y deflection signals for the CRT. The host computer could control the instruction register of the display processor, giving jump and subroutine jump instructions to support segmented display list structures. Subsequent vector display processors exploited hardware to provide matrix multiplication, clipping and vector generation, but the drawback was that the screen required flicker free refreshing, hence, fast processing was required to maximize the number of simultaniously displayable picture elements. Since the hardware was arranged to provide multi-stage processing of picture elements, pipelining was also used to improve performance. Today – many graphics processors provide a refresh buffer to hold the fully transformed picture elements. These are then passed to the vector generator which can run unimpeded by the restrictions associated with transformations and clipping. A double refresh buffer is often installed, enabling one buffer to be used for picture update whilst the other is used for screen refresh by the vector generator. Several other improvements have been made to these buffered vector devices but nevertheless they remain complicated I/O devices resulting in non portable application program development.

1.7.2 Bit Mapped Systems.

The advances described above have brought about very powerful graphics systems, but they are also very expensive and require hardware that occupies many large circuit boards. These systems are very specialised and may never be used in low cost workstation systems. The way in which de-centralised

simulation is used to improve performance is however important, and has been applied by IC manufacturers who have captured a general purpose market place with low cost devices for raster scanning graphics applications.

To support these systems, IC manufacturers have supplied a device which is known as a CRT Controller. The power and functionality of the CRT controller has improved in a similar manner to that of the microprocessor, so that today, much of the processing associated with the graphics system may be offloaded from the workstation microprocessor.

Raster scan systems originally used many Medium Scale Integration(MSI) parts just to generate local and display monitor timing signals. The earliest CRT controllers reduced chip count by integrating the timing control functions onto one device. They were generally programmable to allow flexible configuration of the overall system. A very popular example is the HD6845[14] which is still in use today. This device provided system timings and supported both character oriented and bit mapped output displays.

A system using one of these early CRT controllers either required the host microcomputer, or some other specialised processor, to handle all the update operations on the frame buffer. If the host microprocessor has the task of managing frame buffer operations, then system performance will be degraded. Collisions between frame buffer accesses for drawing and accesses for display refresh further degrade performance. Bit slice elements or a second microprocessor can be used to offload graphic processing from the host computer. The overheads associated with this approach are increased chip count and the time associated with specialised hardware and firmware development.

A new generation of CRT controllers has grown to meet the requirement for

graphics processing with minimum overheads. One of the early "intelligent" devices, the EFCIS EF9365[15] implemented the Bresham scan conversion algorithm to support simple vector drawing. It also contained a character memory, allowing characters to be drawn directly into the frame store with limited scaling and rotation mappings. The frame store was separate from the system memory which gave the EF9365 total control over the arbitration of drawing and refresh memory cycles.

The features that are typically supported by today's CRT Controllers include logical x-y addressing, high level primitives, large frame buffer addressing ranges and the use of First In First Out(FIFO) registers between the host computer and the CRT controller. Frame store addressing in the "Z" direction is also supported to cut the overheads associated with multiple plane and multiple colour systems. Typical devices that have emerged since the EF9365 are the NEC7220[16] which first introduced FIFO command and data transfer registers and the HD63484[17] which employed x-y addressing of high level primitives and "Z" direction addressing of the frame store.

1.8 Graphics Packages and Standards.

Early graphics packages were necessarily targeted at a particular combination of computer and graphics display processor. These packages were used by a sufficiently large proportion of the computer graphics community to become a de-facto standard. Regional de-facto standards also emerged, for example, the GPGS[18] from Norway, GINO-F[19] from England and CORE[20] from America. Although any one of these could have been acceptance as a standard by the official bodies, it was not until 1983 that an international standard for computer graphics could be agreed upon. This standard is called the Graphical Kernel System(GKS)[21]. It represents the first standard for computer graphics

and has been the result of much work from an international group of graphics specialists who have used their experience with many of the earlier de-facto standards to achieve a specification for GKS that encompasses their best concepts and also presents some new ones. The details of GKS are given in chapter 3

Since the acceptance of GKS as an international standard, work has not halted on standardisation for computer graphics and both hardware and software aspects have come under scrutinisation by the standards bodies. The efforts of these standards bodies are motivated by the desire to have a system whereby application programmers requiring computer graphics can address graphical devices in a virtual sense so that they may expect their programs to run on a wide variety of hardware. This portability of software is an obvious advantage, which also brings with it the portability of graphics programmers who, once they have mastered the concepts of a standard, may program using that standard on a wide variety of machines. When moving to a different programming language environment, only the new language binding to GKS must be learnt.

Since CAD application programs require considerable graphical input and output, the adoption of GKS as the graphics application interface for a general purpose CAD workstation is therefore highly desirable.

1.9 Summary.

This chapter has examined the evolution of computer equipment for interactive use, highlighting where the advances have been made that have enabled desk top machines suitable for CAD to be developed. The study has revealed the main hardware and software components that must be integrated

to produce a microprocessor based workstation. It has also shown where the potential bottlenecks may develop and the solutions that exist to overcome them.

The functional components of a general purpose workstation are illustrated in figure 1.11. This system may be logically partitioned into the components associated with a general purpose computing facility, and those associated with its graphical functionality. This study has taken an approach to workstation development which is based on the integration of existing computing systems with custom graphical hardware and software components to complete the necessary workstation environment for CAD.

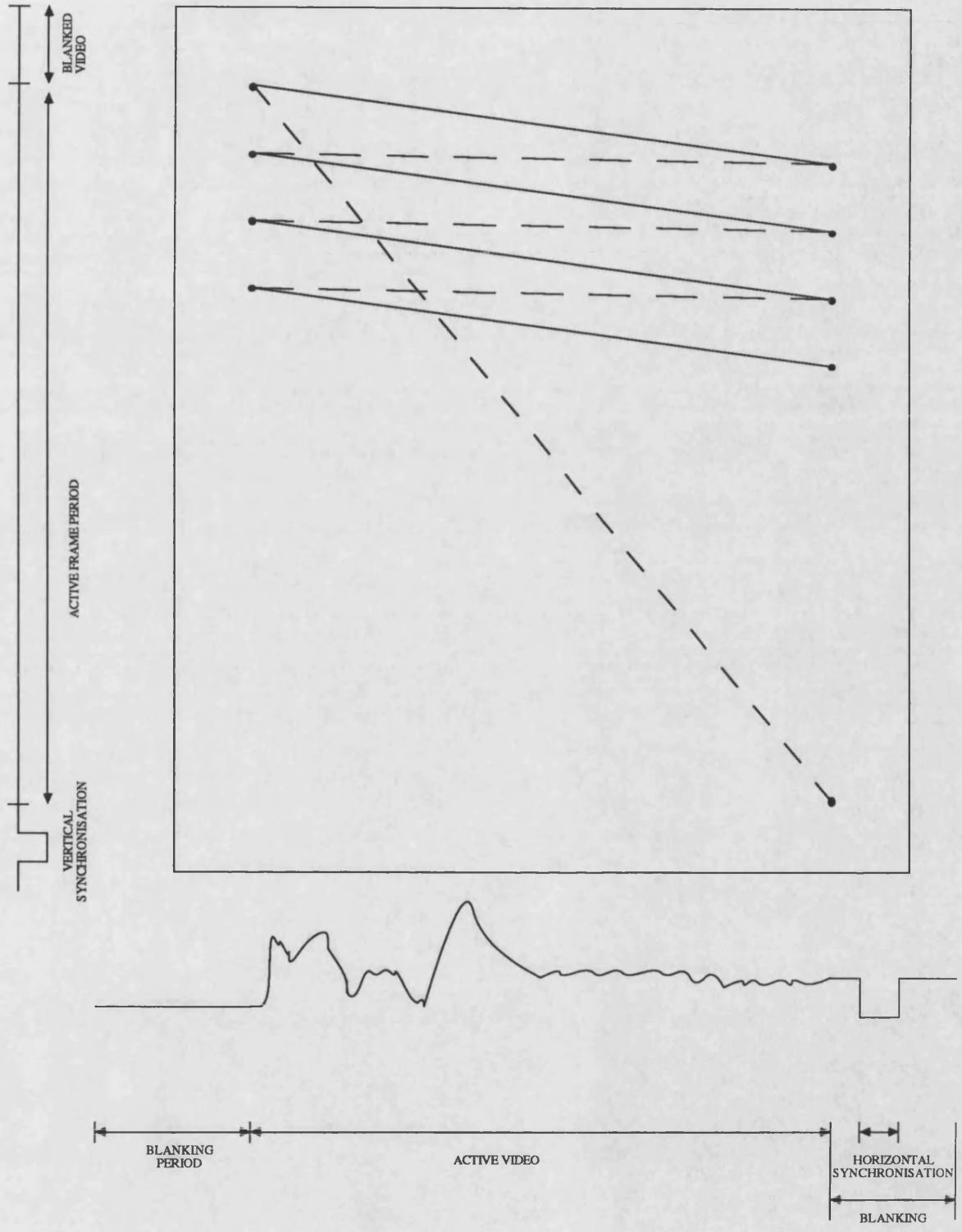


FIGURE 1.1 NON-INTERLACED RASTER SCANNING

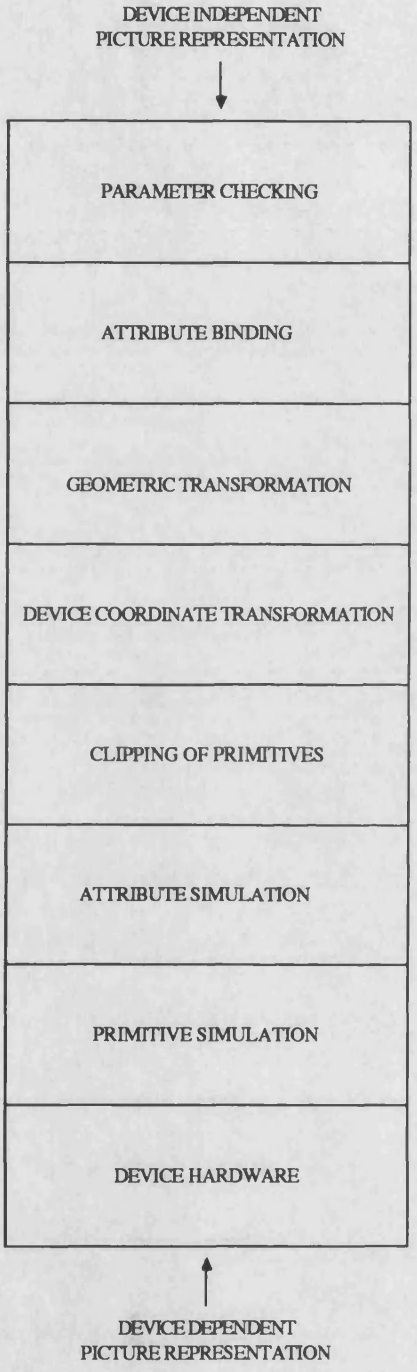


FIGURE 1.2 STAGES IN A TYPICAL VIEWING PIPELINE

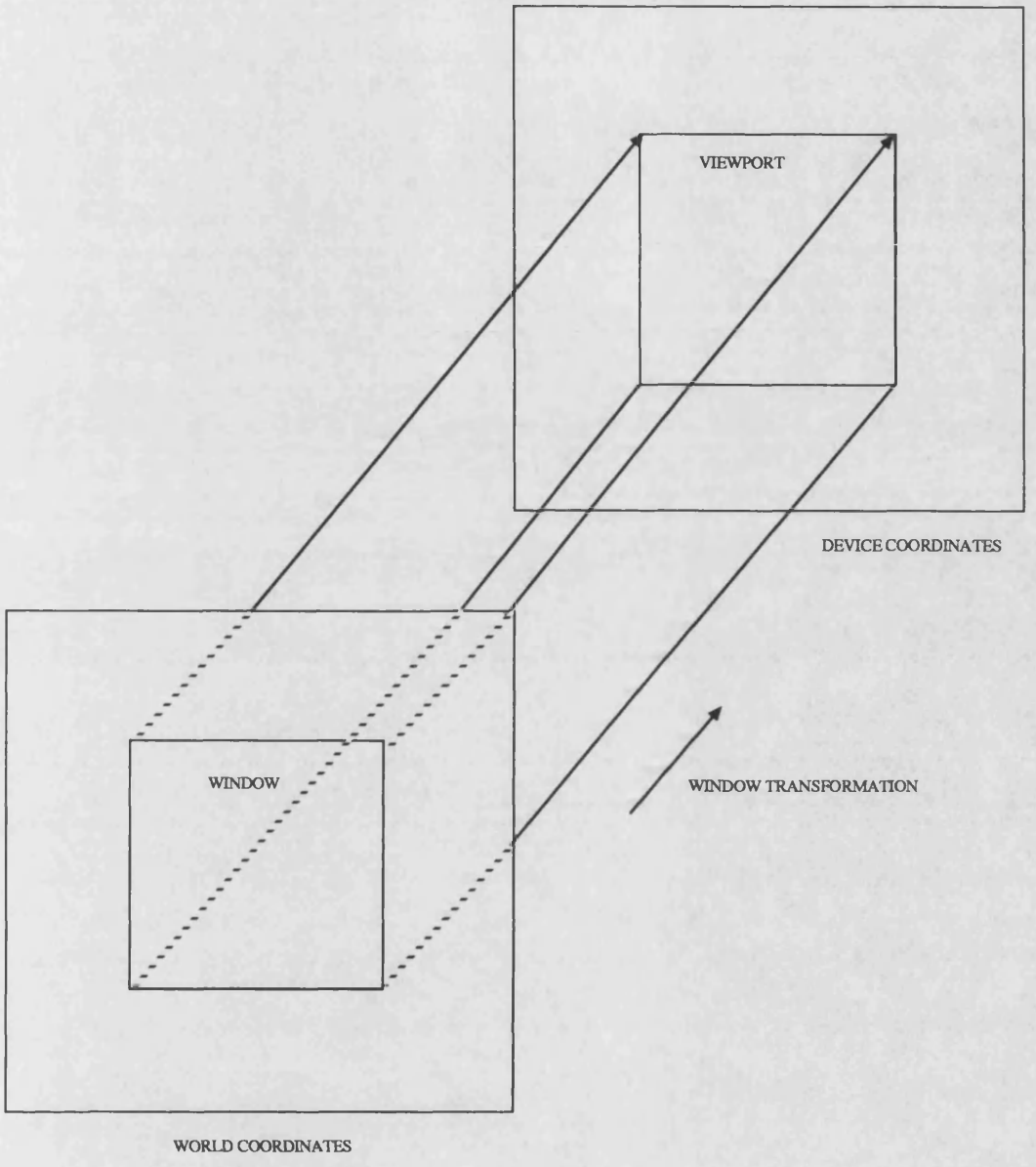
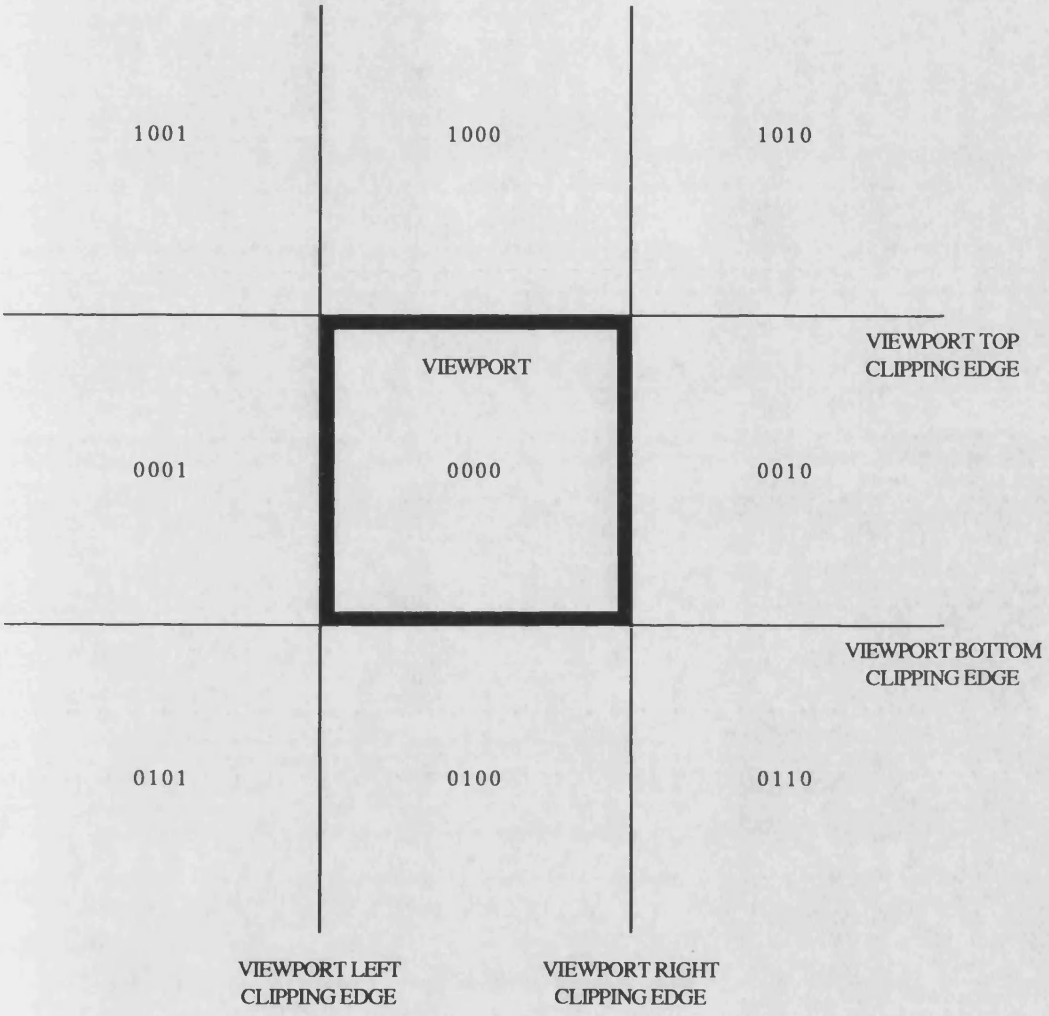


FIGURE 1.3 WINDOW TO VIEWPORT TRANSFORMATION FROM WORLD COORDINATES TO DEVICE COORDINATES



Bit 0	:	point is left of left-hand-edge
Bit 1	:	point is right of right-hand-edge
Bit 2	:	point is below of bottom edge
Bit 3	:	point is above of top edge

FIGURE 1.4 ASSIGNMENT OF OUTCODES IN COHEN AND SUTHERLAND CLIPPING ALGORITHM

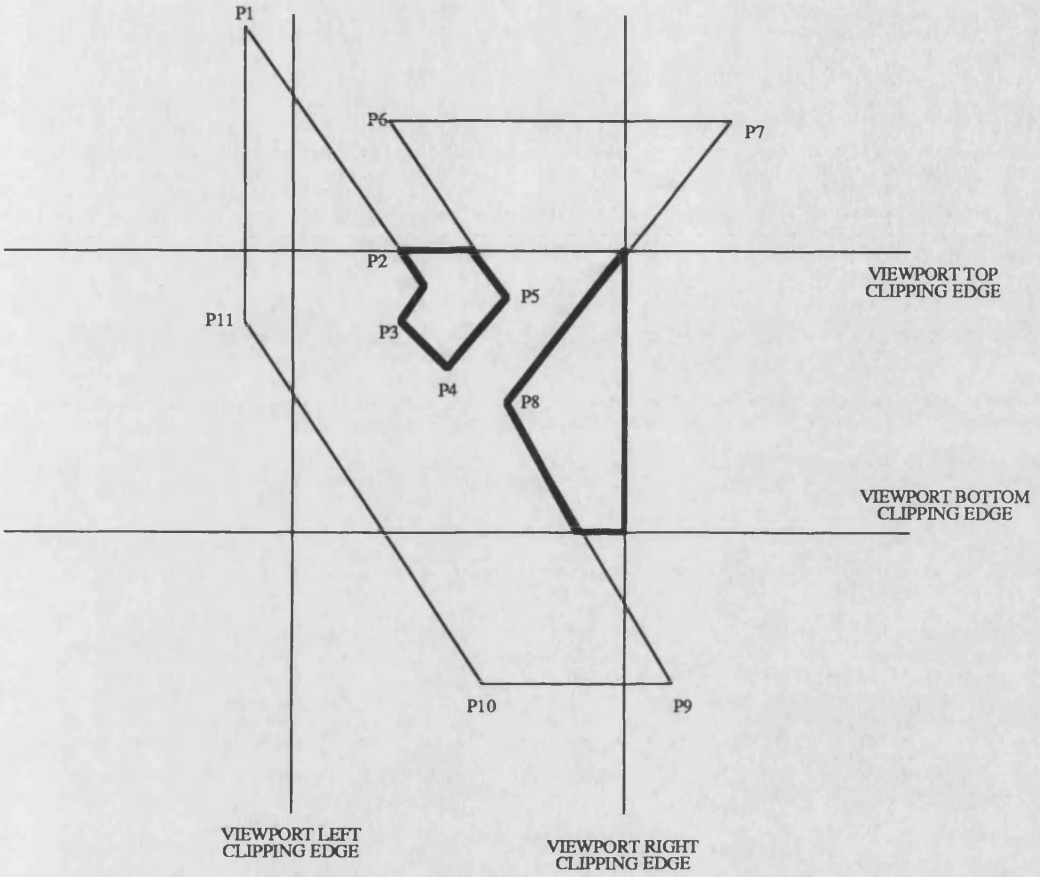


FIGURE 1.5 GENERATION OF DEGENERATE BOUNDRIES WHEN CLIPPING CONVEX POLYGONS

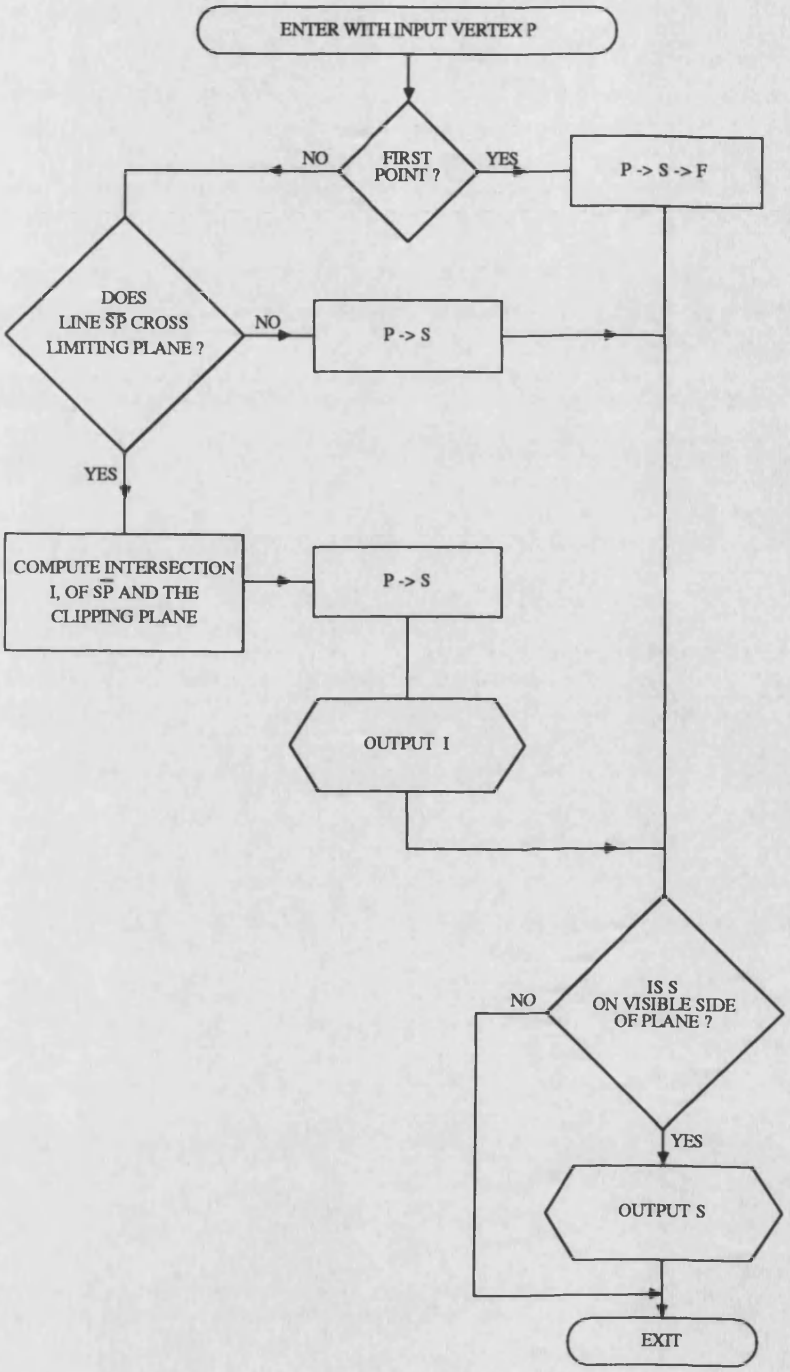


FIGURE 1.6 FLOWCHART FOR STAGE (A) OF THE SUTHERLAND HODGMAN POLYGON CLIPPING ALGORITHM

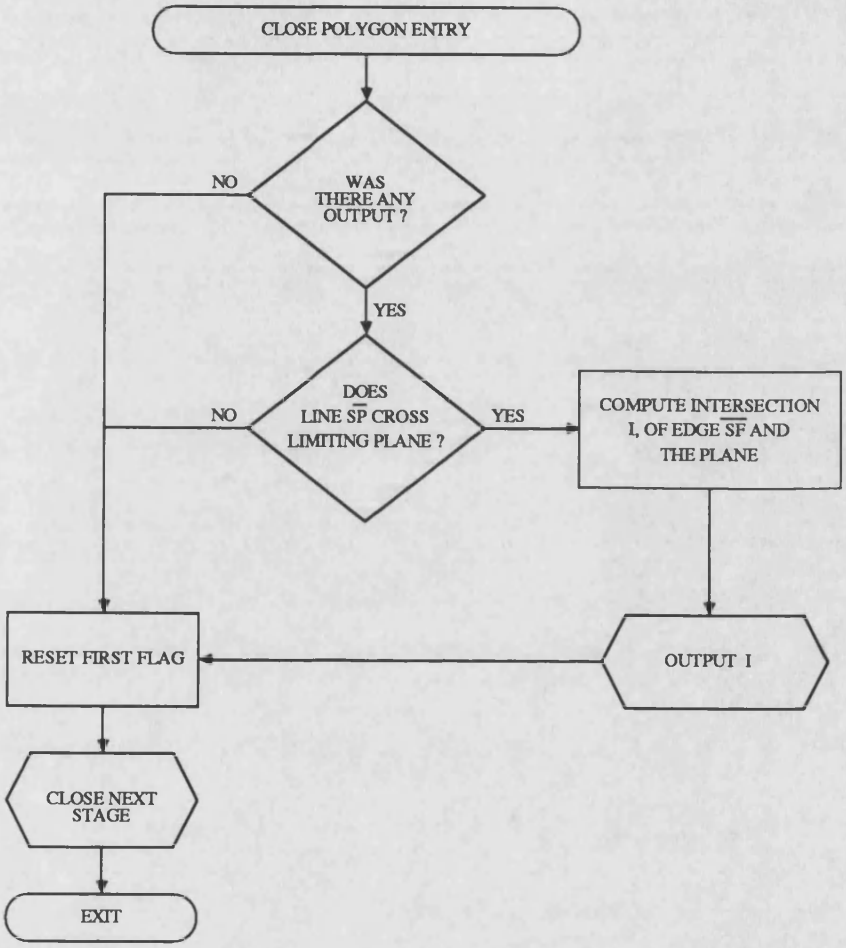


FIGURE 1.7 FLOWCHART FOR STAGE (B) OF THE SUTHERLAND HODGMAN POLYGON CLIPPING ALGORITHM

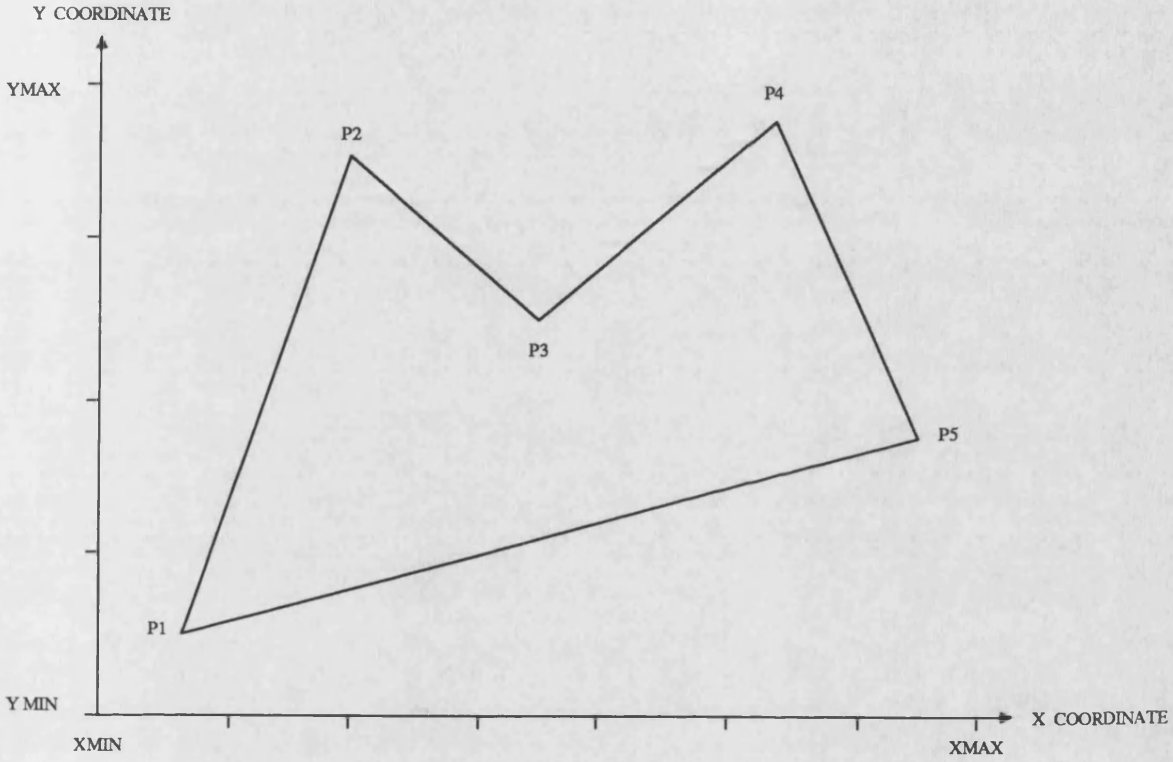


FIGURE 1.8 POLYGON TO BE SCAN CONVERTED

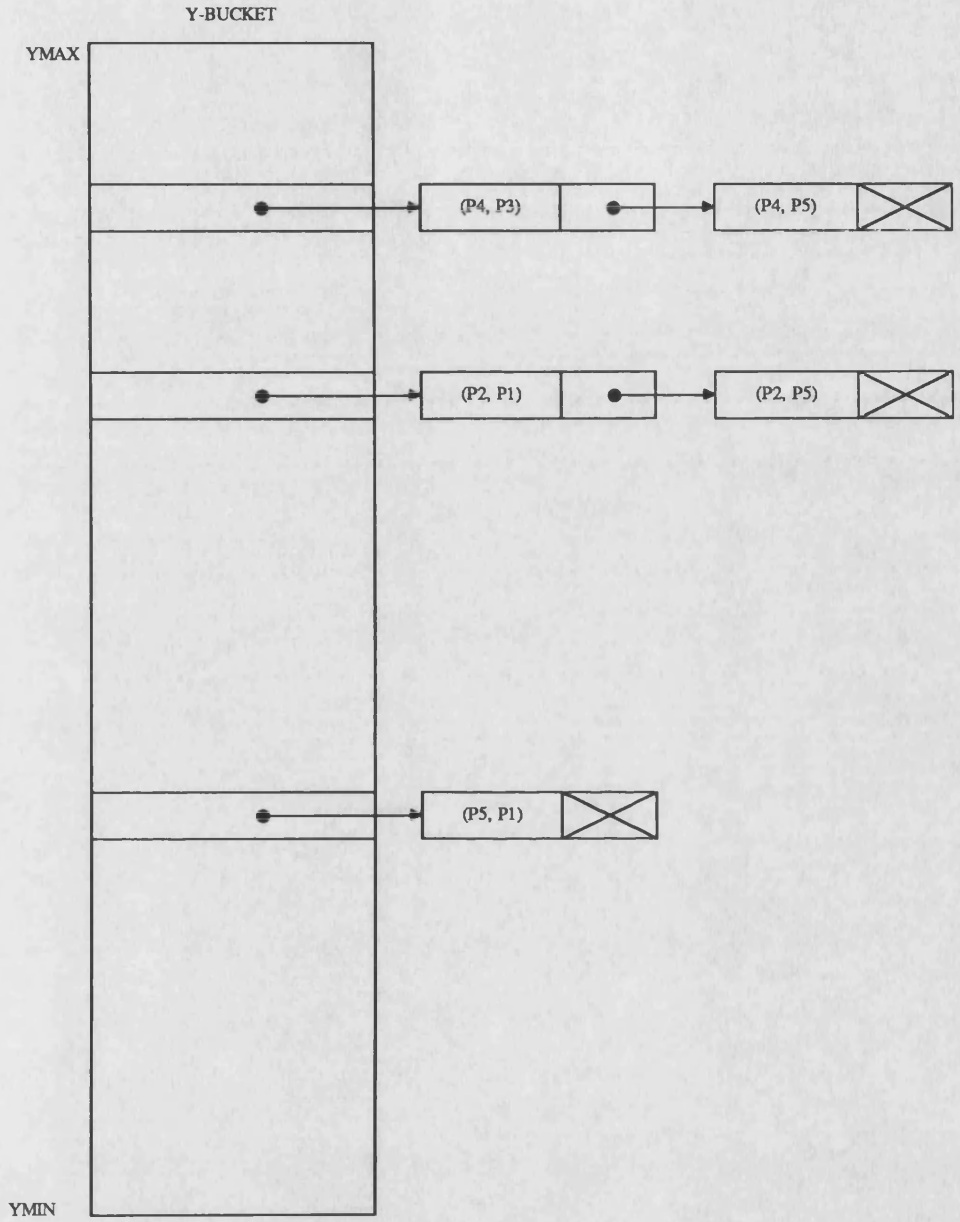
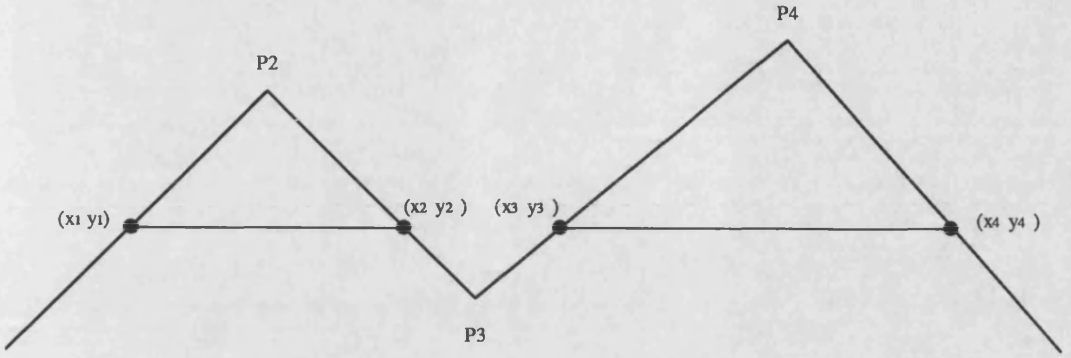


FIGURE 1.9 ENTERING EDGES INTO THE Y BUCKET



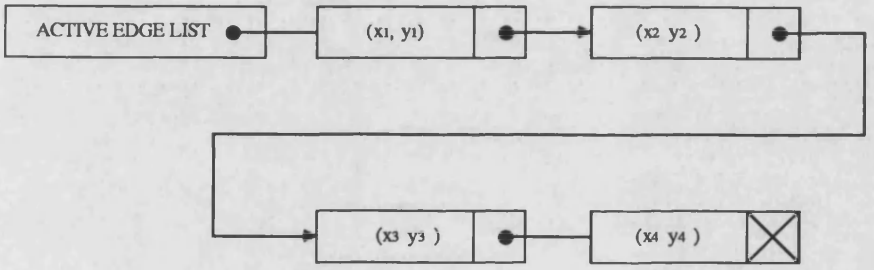


FIGURE 1.10 ACTIVE EDGE LIST AS SCAN CONVERSION IS IN PROGRESS

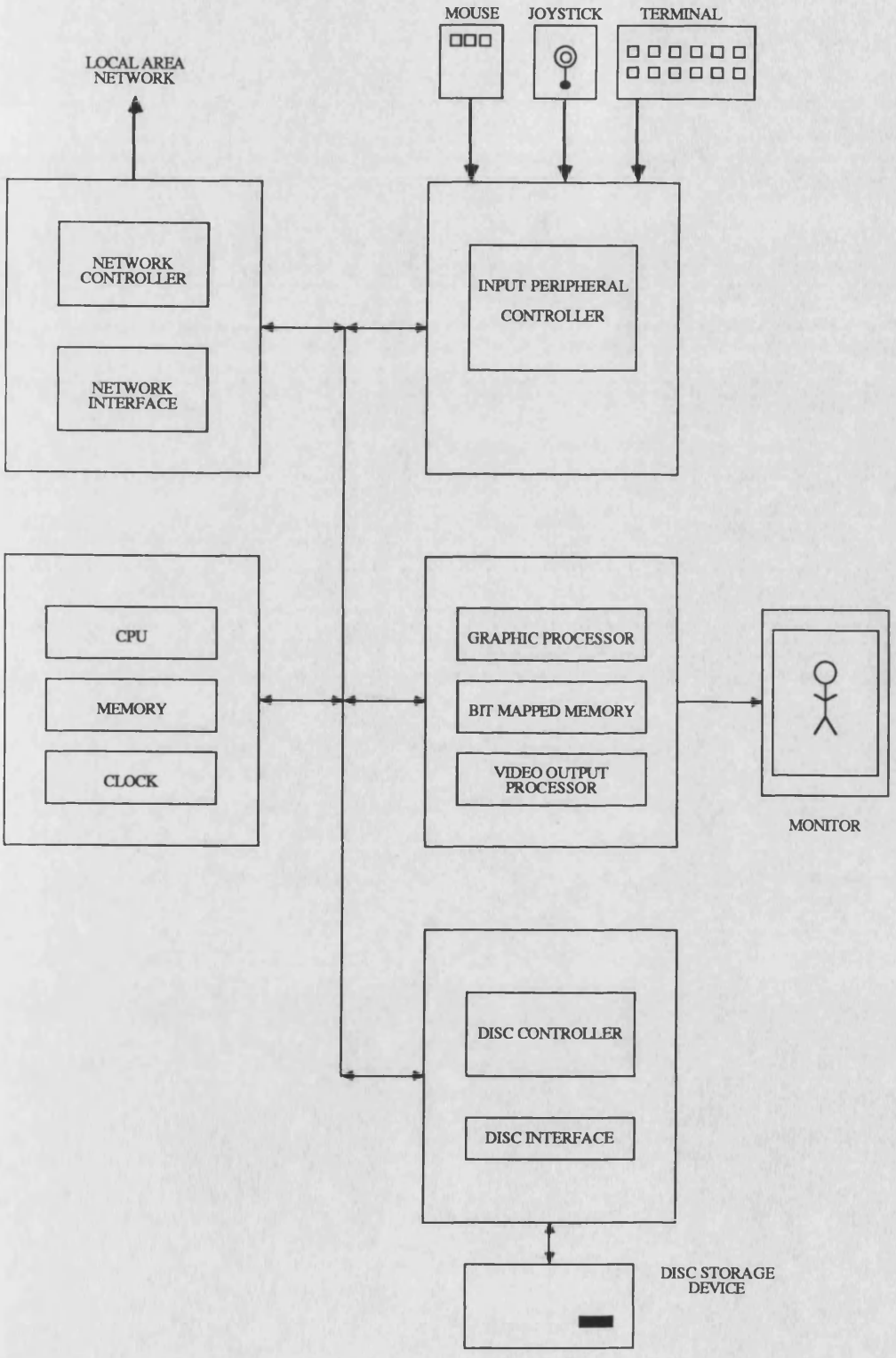


FIGURE 1.11 FUNCTIONAL BLOCK DIAGRAM OF A SINGLE USER CAD WORKSTATION

CHAPTER 2

AN EVALUATION OF COMPUTER SYSTEMS FOR WORKSTATION DEVELOPMENT

2.1 A Microcomputer System for Workstation Integration.

Chapter one highlighted the need for processing power in graphics applications, particularly when the overheads associated with device independent virtual graphics input and output must be overcome. Computer aided design for scientific and engineering applications place stringent requirements on the workstation system if it is to perform satisfactorily. Of particular importance is the response time of the machine to user requests. Factors influencing this parameter will be the speed and power of the processor system, with an emphasis on floating point manipulations, and the speed of data retrieval from secondary storage media which is where the design data bases will typically be stored. The workstation processor must also be capable of directly addressing a large primary memory since CAD applications themselves are typically very large. If all of the program and data areas of a CAD program can be held in memory simultaneously, then the slow process of overlaying program segments will be avoided.

The interfaces supported by a general purpose workstation must, where possible conform to accepted standards. The user interface will be defined by the operating system which should provide powerful facilities to support the underlying hardware and also permit the easy integration of new hardware systems such as those for graphics and networking. The portability of applications and users to a workstation is therefore very dependent upon the choice of operating system. The programmers interface to the graphical

hardware of the workstation should be independent of that hardware. The Graphical Kernel System introduced in chapter 1 should therefore be used to provide this virtual graphics interface. The compilers available on the workstation should support popular scientific programming languages such as FORTRAN 77 and "C"[22] to further enhance the user interface and ensure portability.

The choice of a computer system for a graphics workstation is therefore influenced by all of these factors. This chapter describes the configuration of hardware components, operating system features and programming languages that were evaluated as part of the specification and development of the environment from which a device independent graphics workstation could be constructed. The computer systems should therefore be capable of extension to interactive workstation functionality and be able to support the development and testing of the additional hardware and software components to achieve this functionality.

The microprocessor system chosen for the task of supporting the proposed workstation development was based on the Motorola MC68000 family of devices[23]. The MC68000 is a 16/32 bit microprocessor which has already been proven capable of supporting powerful operating systems[24,25]. Its performance had also been proven in simulation studies of power systems[24]. The computer systems that had been developed for these studies provided the starting point from which to develop the hardware and software that would provide workstation functionality.

2.2 An overview of the Motorola MC68000.

The MC68000, became widely available in 1980 and was the first

microprocessor to provided a true 32 bit internal architecture for its address and data paths. It has 8 data and 8 address registers with one of the address registers being designated as the active stack pointer. Powerful orthogonal addressing modes permit compilers to generate compact and efficient machine code. The instruction set supports arithmetic operations to 8, 16 and 32 bit accuracy. Binary Coded Decimal(BCD) and bit level operators are also available. Externally, 24 address lines and 16 data lines are complemented by other control signals to give a 16Mbyte addressing range with memory fetches on byte, word or long word boundaries. Memory cycles are asynchronous, with hand shaking controlled by a single bi-directional Data Transfer ACKnowlege(DTACK) signal. Figure 2.1 shows memory read and write cycles and the use of the upper and lower data strobe signals in byte and word access cycles. Compatibility with the earlier MC6800 microprocessor devices is maintained by the provision of signals to control synchronous bus cycles.

Programs on the MC68000 run in either supervisor mode, or user mode. Generally systems code will operate in supervisor mode and the stack pointer will be the supervisor stack pointer. Applications programs will run in user mode with the user stack pointer. This distinction is made to afford protection to systems programs by restricting the instruction set that is available to programs running in user mode. Signals are produced by the MC68000 to indicate supervisor program and data accesses and user program and data memory cycles.

The interrupt structure of the MC68000 provides seven maskable levels and one non maskable level of interrupt. Vectored interrupting is implemented such that the interrupting device must supply an eight bit identifier to the MC68000 during its interrupt acknowledge cycle. Multiple interrupting devices on a

single level may be achieved by daisy chaining the processor's interrupt acknowledge signal, this method being used on both the multi-board and single board computer systems. A bus arbitration protocol is defined for the MC68000 to permit other bus masters such as Direct Memory Access(DMA) controllers to drive the system bus. An improved version of the MC68000, the MC68010 which was released in 1982 provided extra instructions and hardware features to support virtual memory/virtual machine systems.

Other devices designed to support the MC68000 include the HD68450 DMA controller[26], the MC68451 Memory Management Unit(MMU)[27], the MC68230 Parallel Interface and Timer(PIT)[28] and the MC68681 Dual Universal Asynchronous Receiver Transmitter(DUART)[29].

2.3 The MC68000 Based Computer System.

The MC68000 computer system was originally based on a multi-board design, using a slotted backplane. During the course of this work a Single Board Computer(SBC) implementation of the system was designed by Dale[30]. A high speed communications link supported multi-processor applications on the multi-board computer system, while the SBC system used the original backplane as its communication medium in a similar manner to that of multi-processor VME[31] systems. The SBC also maintained compatibility with peripheral cards that had been designed for the multi-board system.

With this in mind a multi-processor workstation system was proposed to exploit the concurrency that exists between the GKS graphics processing task and the application task. A tightly coupled multi-processor architecture had been used to solve the aforementioned power system and flight simulation studies and thus provided a possible base on which to develop graphics

hardware and software. Loosely coupled architectures were also evaluated to decide if their use could enhance workstation performance.

The backplane designed for use with the multi-board MC68000 system closely resembled the bus structure of this microprocessor, hence peripherals designed to support the MC68000 and its bus structure were most suited for use with this system. Since its inception, the multi-board MC68000 system had been the test bed for many add-on peripheral cards providing such functions as memory expansion, Winchester disc interface, general purpose input/output, networking, graphics and serial input/output. Documentation for these cards may be found elsewhere[24].

The SBC was designed to be compatible with earlier peripheral cards and hence the existing backplane, whilst also providing the necessary arbitration to permit several single board computers to share a single backplane. The facilities provided by the SBC were chosen from a subset of the multi-board system with the aim of giving a stand alone replacement for this earlier design. The single board computer provides the following facilities :-

1) MC68000 or MC68010 microprocessor, up to 12.5 Mhz.
2) 1 Mbyte Dynamic RAM.
3) HD68450 DMA controller,
4) Two MC68451 MMU.
5) Parallel interface and timer with SASI interface.
6) Floppy disc interface.
7) Dual RS232 serial channels.

The single board computer is attractive for use in workstation applications for several reasons. Firstly, its size to functionality ratio means that the

addition of a networking system and some form of graphical hardware will provide a minimum workstation configuration. The relatively small size of this system permits a desk top machine to be considered. The tight coupling of microprocessor to memory and peripherals allows a 12.5MHz MC68000 part to be used for best speed performance. The single board system was adopted for use in this project for these reasons. Figure 2.2 shows a photograph of the SBC system. The corresponding circuit diagrams are shown in figure 2.3 and 2.4.

2.4 Operating Systems.

The work presented in this dissertation may be logically partitioned into two sections. The first section represents the work that was conducted under the TRIPOS operating system, while the second section, which culminated in the completion of an operational workstation was conducted under the UNIX operating system. The move from TRIPOS to UNIX was made as the requirements for the system changed. The following sections are included to provide an overview of these two operating systems in preparation for the material contained in the following chapters.

2.4.1 The TRIPOS Operating System.

TRIPOS[32] originated from Cambridge University in the mid seventies when the availability of mini and microcomputers with unrestrictive amounts of memory created the desire for portable single user operating systems. Tripos was designed as a single user system, but its multi-tasking functionality can be used to support several simultaneous users. However, no memory management is implemented to protect the operating system data structures from corruption by the actions of a rogue user task. The TRIPOS kernel runs in supervisor mode and tasks run in user mode, therefore, the kernel can at least prevent user tasks

from executing the MC68000 privileged instructions. The small size and relative simplicity of Tripos makes it well suited to dedicated or personal computer applications.

Portability of operating systems was an important issue when TRIPOS was constructed and to maintain this philosophy, the systems programming language, BCPL[33], was used to program the bulk of the TRIPOS operating system utilities. The TRIPOS kernel itself is written in machine code, as are the physical device drivers. This structure makes TRIPOS readily portable.

TRIPOS is a multi-tasking system, consisting in its minimal form of a Command Line Interpreter(CLI) task, a debugging task, a console handler task and a filing system task. A user interacts with the TRIPOS system through the command line interpreter which can initiate the running of system utilities and user programs. A program is run as a co-routine[34] to the CLI, which means that it shares the BCPL Global vector of the CLI. Therefore, the program need not be linked to the standard I/O library of functions as these are all available by reference to items within the global vector. This is also the case for the kernel primitives which are also accessed by reference from the Global Vector. Individual co-routines do, however have, their own stack.

The console handler is responsible for coordinating terminal input and output. A typeahead buffer is implemented together with command line editing. Various escape sequences can be given to the console handler to redirect its logical input and output to any task that is capable of interacting with the operator. By default, the console handler is logically connected to the CLI task, so that all terminal input is routed to the CLI and all output generated by the CLI will appear on the terminal. Another task capable of supporting terminal I/O, and hence connection to the console handler is the debug task.

The filing system task is responsible for managing secondary storage by imposing a suitable data structure on devices such as Winchester discs and floppy discs. The file structure provides a hierachy of directories which may contain a mixture of files and lower level directories. The structure is that of a tree with the root directory at the top. The root directory always resides at some fixed position on the disc allowing any file or sub directory to be traced from it. The file specification is that of the path from the root which is designated by a colon followed by the names comprising the path. Each name is separated by a full stop. A drive name may proceed the colon if there are several mounted devices in use. To avoid wasteful typing of large pathnames, the concept of a current working directory is used, allowing files in the current directory to be accessed simply by using their names. When a disc is mounted for reading and writing, a low priority task, called the restart task, is created which checks the consistency of the disc and sets the time and date to that found on the most recently created or modified file.

The debug task provides a means of monitoring or modifying the program and data areas of the run time kernel, the device drivers, and the resident tasks. It can run in two modes, either as a TRIPOS task or in a stand alone mode where system interrupts are turned off and debug communicates with the user terminal directly. The later mode is entered after an exception occurs or a TRAP instruction is executed. The TRIPOS routine abort causes a TRAP instruction to be executed and is used to signal an unexpected event from which debug may be used to clear up or examine the cause.

TRIPOS tasks have allocated to them a priority which is used by the scheduler when the currently running task halts and another task must be started. The task with the highest priority that is then able to be run will be

started, but, if no task is available then the idle task, which is always guaranteed to be ready to run, will start. Halted tasks may simply be waiting for something to happen or may have been interrupted and are waiting to be resumed.

The only fixed TRIPOS data structure that has a fixed position in memory is the root node, shown in figure 2.5. All other TRIPOS data structures can be reached from the root node by following the correct chain of pointers. Tasks are managed by TRIPOS through the use of a task table which has an entry for each task that points to a per task data structure, called a Task Control Block(TCB). The TCB contains information about the priority, state and location of program segments associated with the tasks. A path through the root node also accesses the device table which is a table of pointers to information about the currently loaded logical devices in the system. Devices may be dynamically loaded if their use is not required on a permanent basis.

A logical TRIPOS device consists of two parts, the Device Control Block(DCB) and the actual driver code body, both of which are written in machine code. Figure 2.6 illustrates how the device table is chained to the device DCB which is then used to access the driver routines. There are no back pointers from the driver code section so several physical devices may share the same driver code. However, a separate DCB must be allocated to each physical device since the DCB must maintain its own private work queue for each device.

A device driver contains five machine code subroutines that can be called from outside, these being, Init, Uninit, Start, Stop and Int. Init is called when the device is created, its function being to set up the call addresses of the Start Stop and Int routines in the DCB and to provide any necessary action on the physical device. Uninit is called when the device is to be deleted and, in most

instances, will do nothing. Start and Stop are used, respectively, to initiate or cancel any work that has been sent to the driver. Finally, the Int routine provides a response to an interrupt from the physical device to indicate completion or failure of the current action. For the MC68000 implementation of TRIPOS, where vectored interrupts are available, the vector is set to jump to a location within the DCB for the particular device which then contains a jump to the interrupt routine in the driver. This chain of events is shown in figure 2.7.

In Tripos, the mechanism of communication between tasks and devices is identical, being based on a packet passing protocol controlled by the kernel primitives Qpkt and Taskwait[35]. The messages are in the form of a vector for which two locations have a predefined meaning. One of these is used to identify the task or driver to which the packet is to be sent, the other is a flag to indicate its usage, being initially set to indicate that it is not in use. Tasks are identified by positive integers and devices by negative integers, with the exception of -1 which is always used for the system clock. The Qpkt-Taskwait method of communication is very efficient since the single address feature of TRIPOS allows packets to be passed by reference.

A multi-processor extension has been added to TRIPOS to enable tasks on different processors to communicate with each other using the Qpkt-Taskwait primitives. The SBC is used to support this multi-tasking, multiprocessor operating system, and therefore it could be used to implement a tightly coupled multi-processor workstation.

2.4.2 The BCPL Programming Language.

BCPL is the TRIPOS system language. Its run time environment comprises a

stack and a vector for global variables. It is a low level block structured language. It is a typeless language with fixed length storage cells, leaving the task of data abstraction and hence the interpretation of variables use up to the programmer. Since the user is free to attach his own abstract type definition to any variable, care is needed to avoid coding nonsensical operations such as adding a bit pattern representing a floating point number to a fixed point integer. With careful use this feature can be put to good advantage by allowing dynamic data typing within data structures such as vectors, and matrices. A rich set of conditional control functions are provided to enhance readability and eliminate the need for the use of GOTO statements. Memory indirection is supported by the provision of operators that can interpret and manipulate the addresses of objects. BCPL is very suitable for testing and programming customised hardware since values may be assigned to machine addresses directly from the programming language. Unlike languages such as FORTRAN, BCPL does not have a set of input and output functions defined as part of the language syntax, but instead, a run time library of procedures and functions provide this facility together with most of the standard mathematical functions. Tripos is largely written in BCPL and hence all the system commands are directly accessible from the language, enabling powerful and efficient applications to be written. These functions are again accessed via the standard BCPL library.

2.4.3 The Unix Operating System.

The Unix operating system[36] is a true multi-user, multi-tasking operating system built on the philosophy of providing a comprehensive, yet non complex set of functions for the user. A rule of one good utility to fulfil a requirement rather than many with overlapping functionality has led to the clean

appearance of Unix.

UNIX consists of two majour components, the operating system proper (the kernel) and the applications software that it supports. The first implementations of the UNIX kernel were totally written in machine code, but later a large proportion was re-written using a descendent of the BCPL programming language, called "C". Today the UNIX kernel comprises around ten thousand lines of "C" source code and one thousand lines of assembly code. The "C" run time enviroment is very simple, requiring only a stack. UNIX is therefore highly portable and this fact is emphasised by the number of mainframe, mini and microcomputer systems that now run UNIX. The assembly code that remains in the kernel is necessary to implement features not available in C and also to make time critical functions more efficient.

The UNIX kernel is a tool which may be used to configure a system for a given need. A good example of this is a command language interpreter called the shell[37]. The kernel interface consists of two main functional components, the filesystem and the process control interface. A very wide range of utilities have been built on top off this interface to aid the applications programmer. These include compilers, assemblers and source code management schemes. Hence, the UNIX environment would be a very suitable platform on which to develop a workstation that would support the development of CAD applications.

The filesystem maps onto the underlying computers peripheral hardware. This I/O interface is unique in its technique of dealing with the large and varied peripherals that may be attached to a computer system. The I/O interface insulates the user from differences between computer peripherals. All physical devices, including disks, terminals and memory are part of the file system and all I/O calls to them go through the file system via a set of system calls. The

I/O calls are open, close, read, write and seek. This interface allows an application to be unaware of the physical device it is communicating with, hence freeing it from implementing complex device specific functions. UNIX applications are therefore machine independent and hence portability is maintained.

The process control interface gives control of the multiprogramming enviroment and supports time sharing applications. This interface governs the creation, termination and communication of user processes. In addition, it provides dynamic allocation of resources such as memory to running processes. The kernel does not contain any unnecessary or over complex functions so it plays very little part in the outward appearance of the applications enviroment and does not support any real-time control features, such as user assignable program priorities. The process control interface allows the outward appearance of the operating system to be governed by a set of specialised user programs.

An application program can request services from the operating system by way of system calls. These calls execute a part of the operating system code and then pass the results back to the application. Hence, an application runs both user code and system code and forms part of an execution enviroment called a process. Each UNIX process may be regarded as a virtual machine which can support both user application programs and subroutines which can be called to obtain I/O and process control services. If a kernel subroutine is executing, the system is said to be running in system or supervisor mode, and if a user subroutine is executing, then the machine is in user mode. In the UNIX realisation of the process virtual machine, the system mode maps onto a re-entrant invocation of the UNIX kernel and the user mode maps onto an invocation of the users program image. The UNIX system maintains a track of

the read only text segments that are associated with each process in a text table which contains entries for the segments position in primary and secondary memory and a use count of the segment that is loaded. This latter parameter is used to allow code sharing between processes. When the use count for a segment is zero, its table entries and primary and secondary memory allocation are freed. The stack of a user data segment has a boundary that is increased automatically when a page fault occurs. The other data area may be increased by explicit system calls.

Each user process has a fixed size system data segment associated with it. This data segment holds all the information required by the system when a process is active, such as the saved processor registers, open file descriptors, accounting information, a scratch data area and the stack that is to be used for the system phase of the process. This data segment is not addressable from the user process and is hence protected.

When a process is not active the system uses an entry in the process control table to determine such parameters as the process name, the location of segments associated with the process, and scheduling information that may be used to determine when the process will be re-started. The process table entry is allocated when the process is created and freed when it terminates.

New processes are generated by the Fork system function, which generates an identical process referred to as the child of the original parent process. Identifiers are returned to the parent and the child so that they may ascertain their relationship to each other. The child inherits its parents environment, resulting in the sharing of open files and other system resources. A parent process may wait for its children to die before recommencing. A process may be made to run a different program by overlaying itself with the text and data

segments of that program by using the Exec system call which maintains the environment after the call, hence leaving open files for the new program.

The shell program uses the Fork and Exec mechanism to run commands issued by the user. Firstly, Fork is used to produce two processes. The child process then calls Exec to overlay itself with the required program. The parent process is then made to wait until the child dies, control then being passed back to the shell.

To enable many processes to run simultaneously, which may require more primary memory than is available, a technique of swapping is employed to move the major data segments of a process between primary and secondary memory(usually a disc device). A system process called the swap process will move the data segments of a process in primary memory onto secondary memory if another process is waiting to be swapped in, or if the process grows and there is no free primary memory to allocate to it.

Since all physical devices are mapped into the file system by use of special devices, drivers must be written to support the I/O system calls for each required device. There are two classes of device, known as block and character which support block oriented transfers of data typical in disc or tape transfers and random length data transfers such as terminal I/O. Block and character devices are also referred to as structured and unstructured devices respectively. Devices of both classes are named by a major and minor device number, with the major device number selecting which driver is to be used and the minor number is passed to the driver to select a sub device such as one particular disc drive that may be connected to a controller.

The use of the array of entry points (configuration table) as the only link

between the system code and the device drivers is very important and allows systems to be easily configured and new device drivers to be added. The configuration table is usually created automatically by a program that reads the system parts list. Majour device numbers for character and block devices index separate tables and so may overlap if required.

The device driver may supply a subset or all of the following system calls open, create, read, write, close and ioctl for a particular device.

Character drivers will be considered here as they are pertinent to material contained in following chapters. The open routine is called each time the file is opened with the full device number as argument. The second argument is a flag which is non zero if the file is to be written to. The close routine is called only when all processes that have access to the file have requested a close action. The first argument is the device number, the second is non zero if the final closing process opened the file for writing. The write routine is passed the driver number as argument, and uses parameters from the per-user data area called u.u_count, a count of the number of bytes to transfer to the device and u.u_base which is a pointer to the data area to be passed to the physical device. Various system calls are provided to pass data from the users data area to the device or to some internal kernel buffer. The read routine is similar to write and data is passed to the users data area as specified by u.u_base.

Interrupts-time routines are entered when the device signals an interrupt. After the interrupt has been processed, a return from the interrupt handler will return from the interrupt itself. The sleep-wake-up mechanism will allow other processes to run whilst the handler is waiting for an event. It is actioned by the calls Sleep with arguments event and priority, and the call Wakeup, with event as its argument. There are also functions to set the interrupt

priority mask to prevent unwanted interrupts arriving at critical moments.

2.4.4 The C Programming Language.

The C programming language is the systems programming language for UNIX and hence provides the same powerful link to the operating system facilities as BCPL does for TRIPOS. Unlike TRIPOS, device drivers are also written in C under UNIX. Many of the features of C are modelled on BCPL, but it is not syntactically similar to it. The C run time environment is very simple requiring only a stack. C is a typed language providing the base types of characters, several sizes of integers and floating point types. A higher level of data abstraction is provided by the addition of aggregate types known as structures. Typing is not as strict as in languages such as PASCAL, but where checking is required, an additional utility called LINT may be used to report inconsistances in the C source code. Memory indirection is supported through the use of pointers and operators on address types, but because user processes run in a virtual rather than the physical address space of the machine, physical addresses may not be readily addressed in the same manner as BCPL. A library of utilities is supplied for input and output of character oriented data from user processes.

Devices drivers for the special graphical hardware of a workstation will, for efficiency, be written in C, and these are examined in detail in chapter 6. However, programming CAD applications in C may not be the best approach. For scientific applications, FORTRAN is the predominant language, and hence a compiler for this language is supported in the UNIX environment. If applications require access to libraries of functions that have been written in FORTRAN such as a graphics library, then they may do so by complying to the procedure calling sequence defined for the FORTRAN UNIX compiler.

2.5 Summary.

TRIPOS provides a good enviroment for hardware development due to its low level systems support and the BCPL programming language. Its simplicity and straight forward handling of hardware makes debugging easy and test software quick to implement. Loadable drivers make the development of new drivers very quick because a system rebuild is not necessary. The facilities available under TRIPOS are however limited, it is not a well supported operating system and there are few languages available.

The UNIX operating system is fast becoming an industry standard amongst commercial workstation manufacturers. It is highly portable and provides a uniform user interface. It also supports a wide variety of commercial CAD software and compilers, hence, UNIX was chosen as the operating system upon which to base the final workstation development.

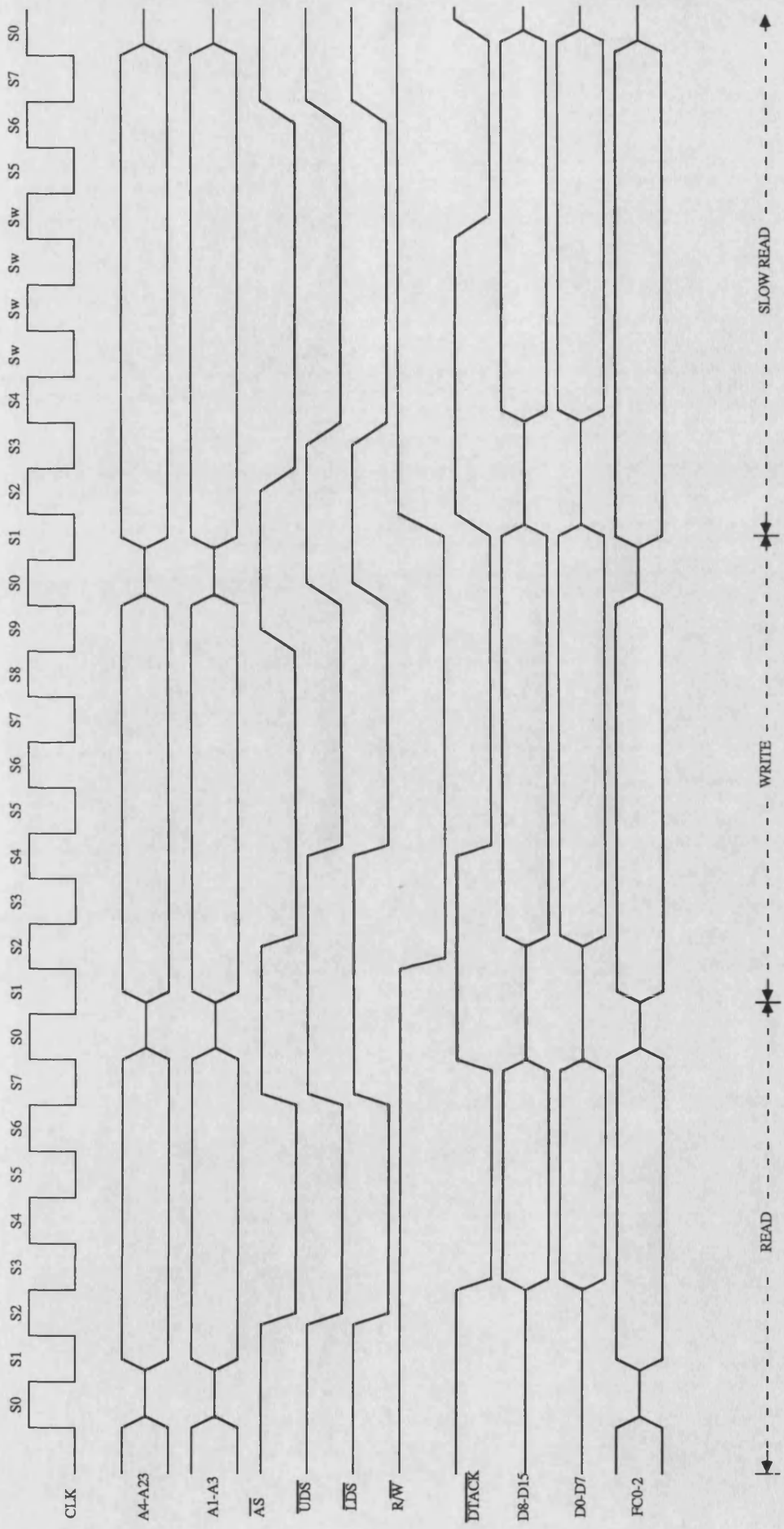


FIGURE 2.1 MC68000 WORD READ AND WRITE CYCLE TIMING DIAGRAM

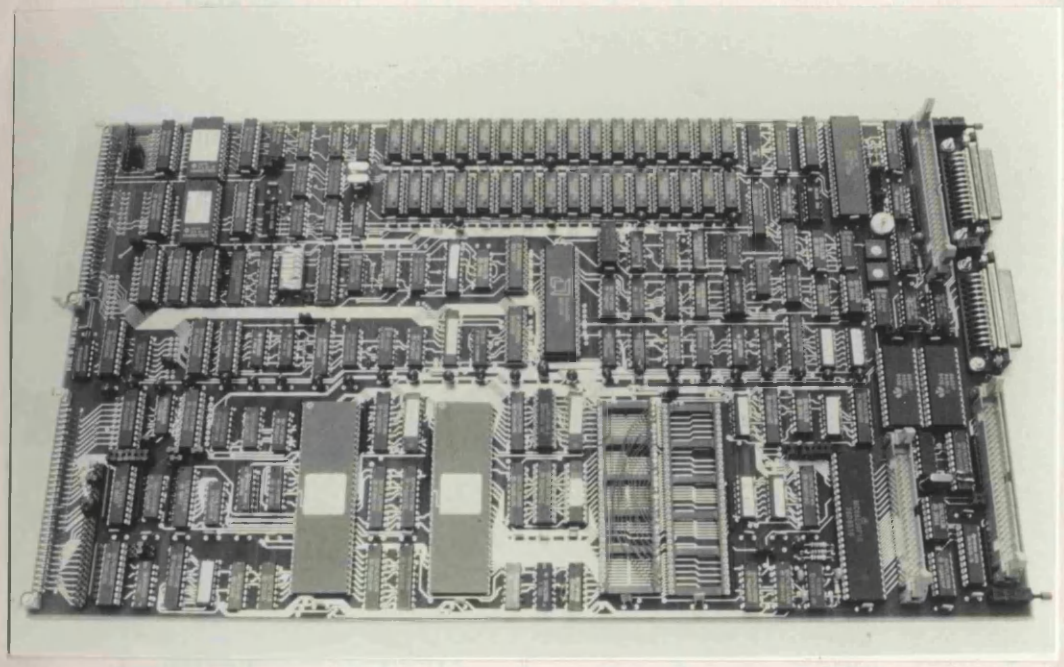
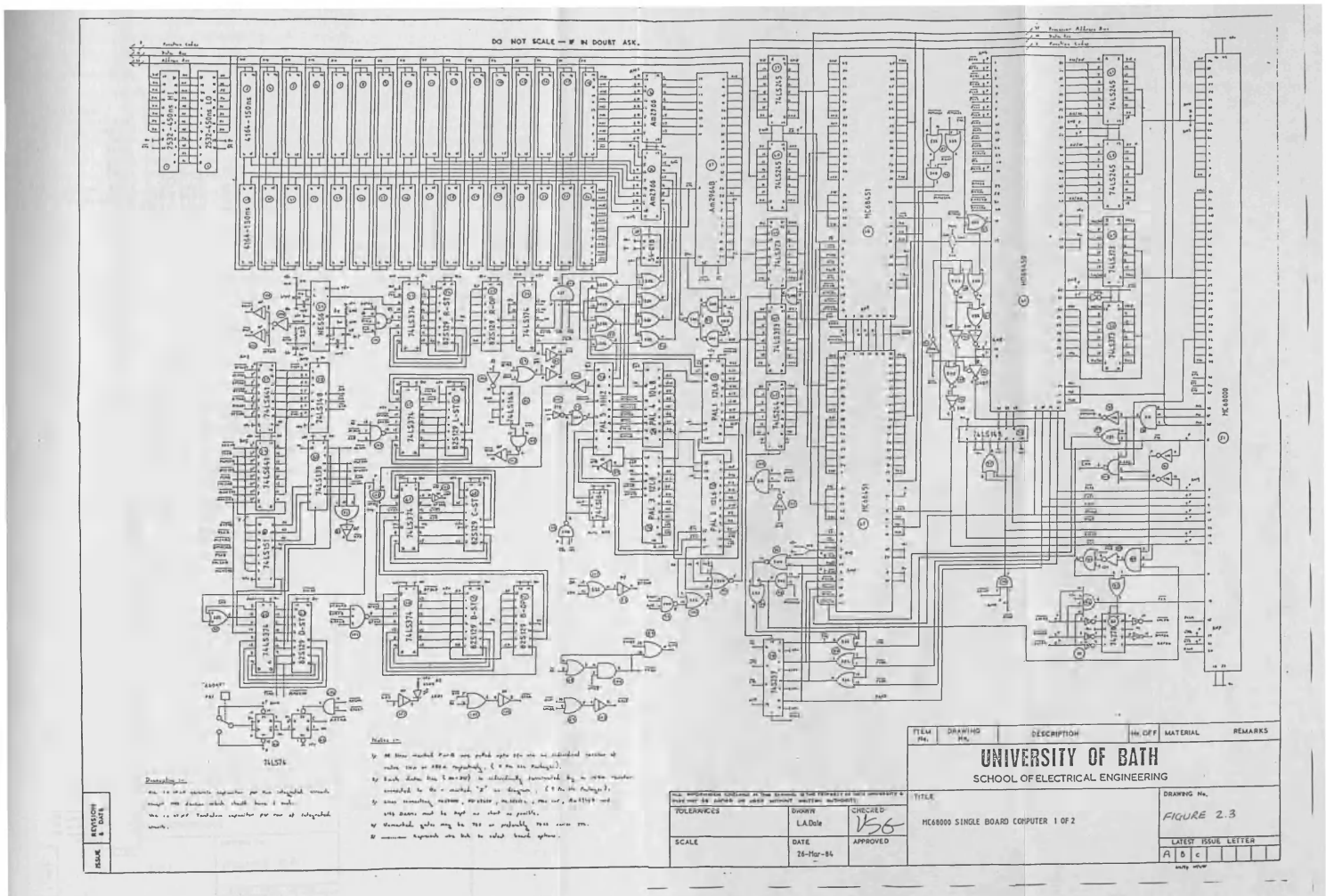
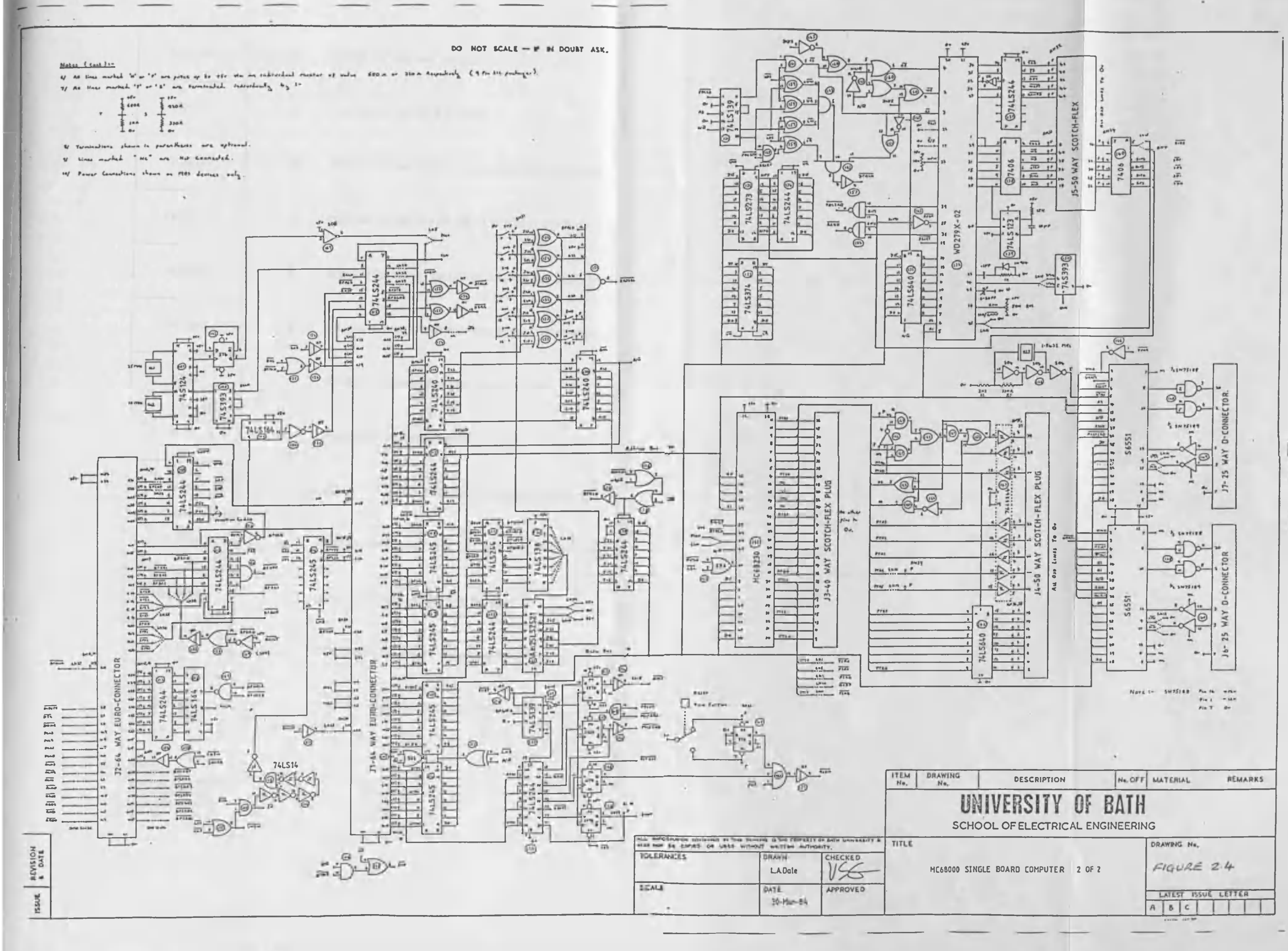


FIGURE 2.2 A FULLY POPULATED SINGLE BOARD COMPUTER CIRCUIT CARD


DO NOT SCALE — IF IN DOUBT ASK.
Decoupling :-
Notes :-
UNIVERSITY OF BATH
SCHOOL OF ELECTRICAL ENGINEERING
MC68000 SINGLE BOARD COMPUTER 1 OF 2
FIGURE 2.3
ITEM No.
DRAWING No.
DESCRIPTION
No. OFF
MATERIAL
REMARKS
TOLERANCES
DRAWN
L.A.Dale
CHECKED
SCALE
DATE
26-Mar-84
APPROVED
ISSUE
REVISION & DATE
LATEST ISSUE LETTER



ITEM No.	DRAWING No.	DESCRIPTION	No. OFF	MATERIAL	REMARKS

UNIVERSITY OF BATH
SCHOOL OF ELECTRICAL ENGINEERING

TITLE: MC68000 SINGLE BOARD COMPUTER 2 OF 2

DRAWING No. FIGURE 2.4

TOLERANCES | DRAWN: L.A.Dale | CHECKED
SCALE | DATE: 30-Mar-84 | APPROVED

LATEST ISSUE LETTER: A B C

ISSUE | REVISION & DATE

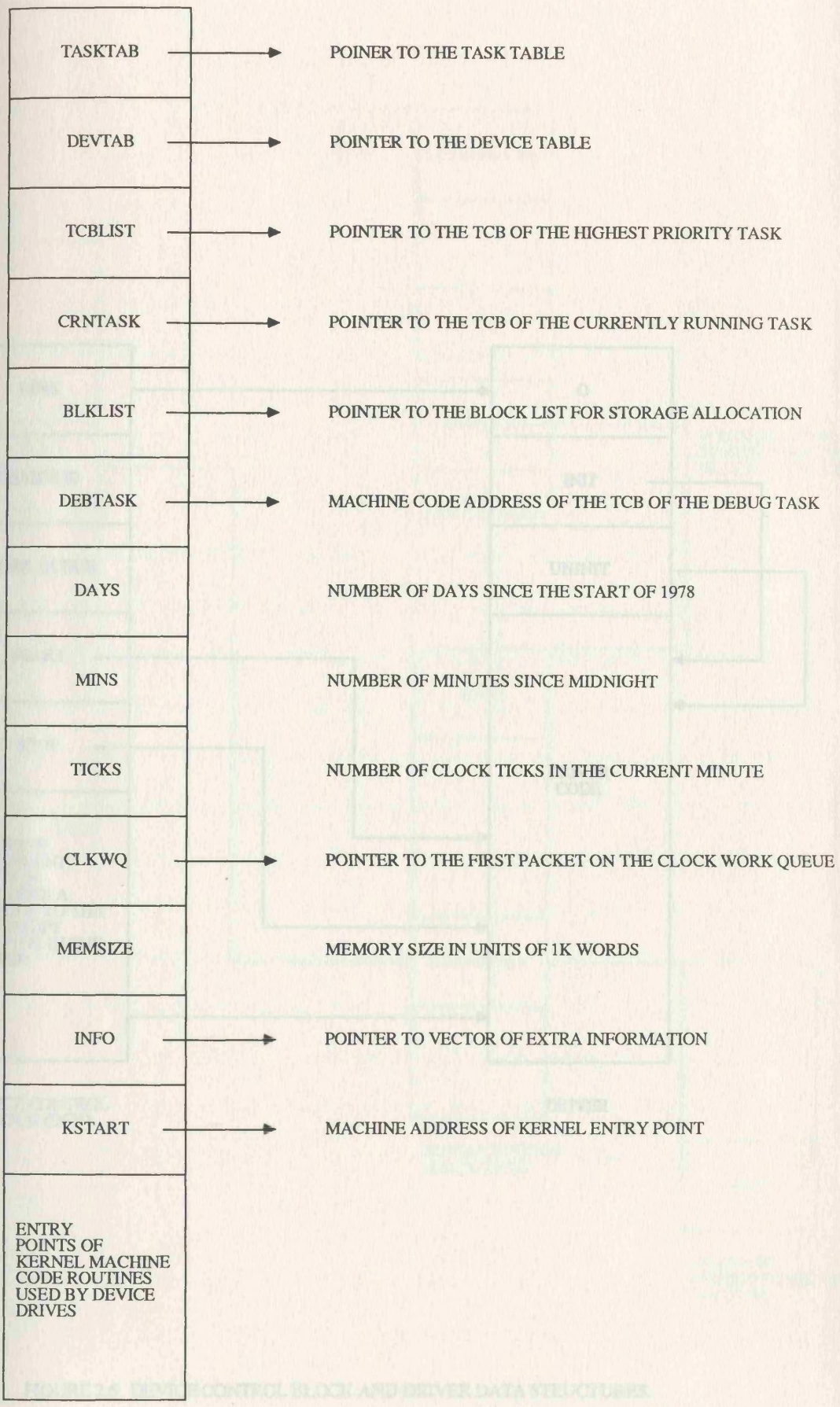


FIGURE 2.5 THE TRIPOS ROOT NODE

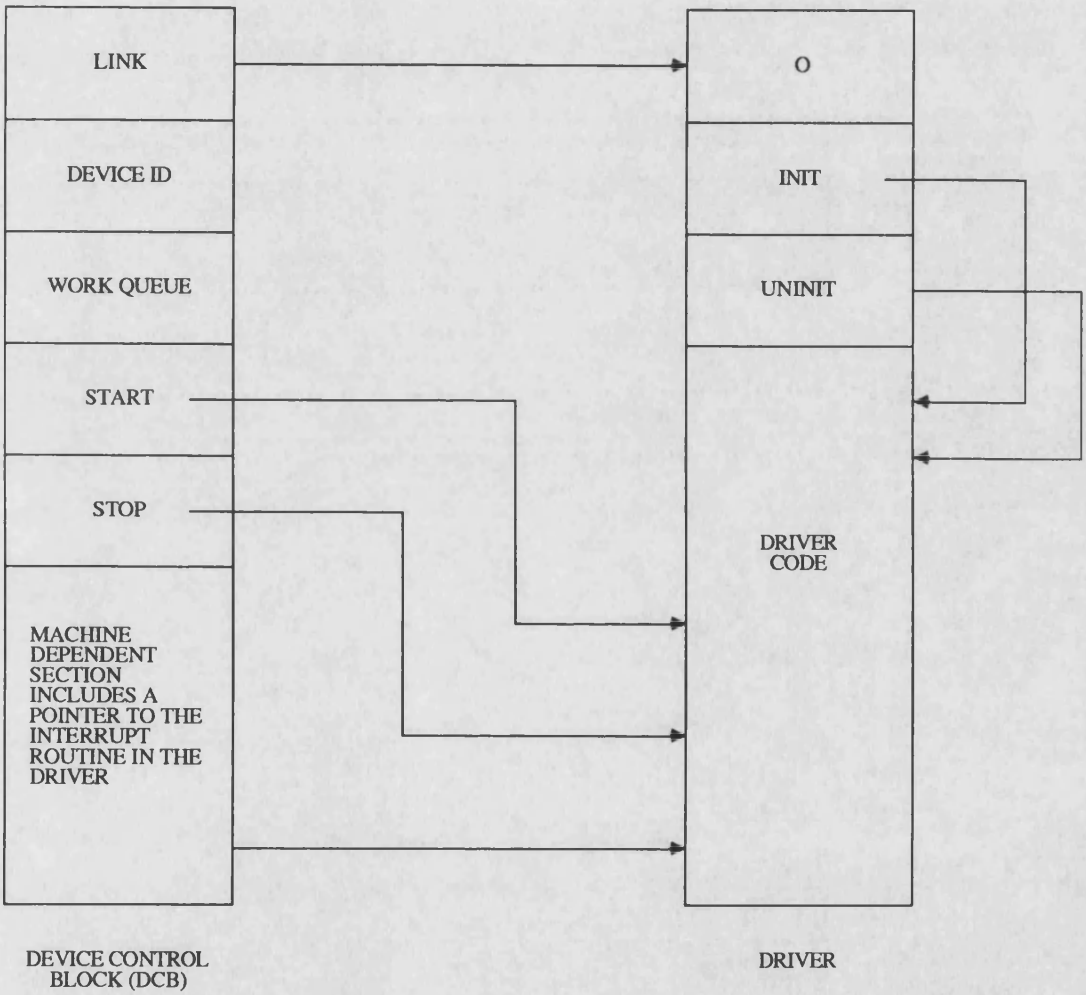


FIGURE 2.6 DEVICE CONTROL BLOCK AND DRIVER DATA STRUCTURES.

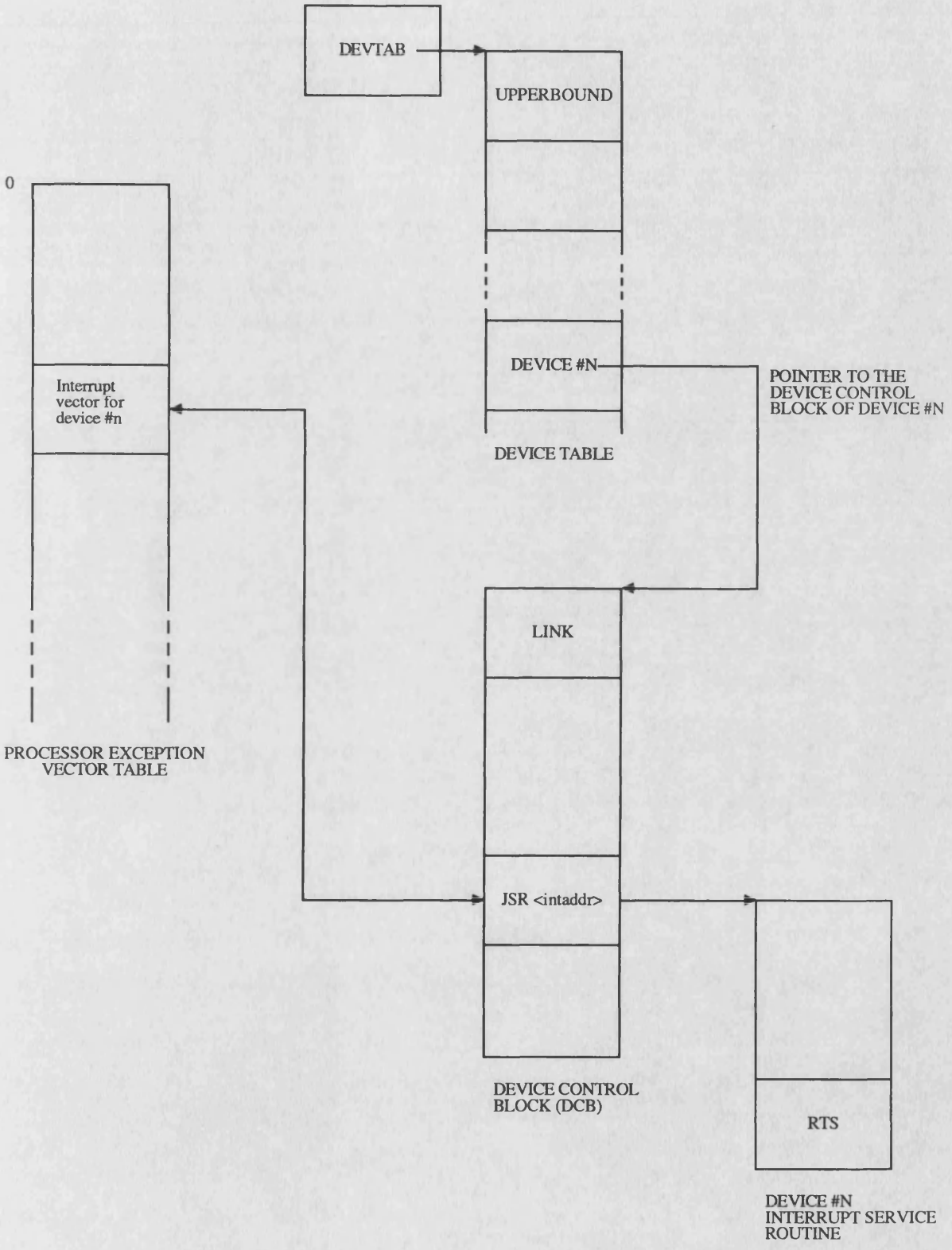


FIGURE 2.7. ASSOCIATION OF INTERRUPT VECTORS WITH DEVICE INTERRUPT SERVICE ROUTINES

CHAPTER 3

THE GRAPHICAL KERNEL SYSTEM

3.1 Introduction.

The presentation of information in graphical form is crucial to the design of a powerful man machine interface, and hence, a workstation for Computer Aided Design(CAD) must provide graphics facilities for picture composition and manipulation. The application interface to such a graphics system should conform to a well accepted standard in order to make the workstation as general purpose as possible. The potential usefulness of the workstation will be achieved if applications can be easily ported onto the system without requiring lengthy modifications. From within the application, a uniform interface to the physical hardware comprising the system will result in a more compact and concise handling of devices since the same code may be used to drive hardware with widely differing features. The graphics system must offer facilities that save the application programmer from becoming concerned with the physical details of graphics hardware and save him from implementing special utilities for picture manipulation or device handling. Any future enhancements of the applications code to drive different or more powerful devices will be straight forward since the uniform interface will hide any of the hardware peculiarities. The conceptual structure of the graphics system must be founded on well established techniques and methods to prevent lengthy retraining and familiarisation on the part of the application programmer.

The Graphical Kernel System(GKS) was adopted as a standard for computer graphics in 1983 by the International Standards Organisation(ISO). It was the

first graphics system to achieve such a status and offered the opportunity to produce a graphics application interface for the CAD programmer that offered device and machine independency. The underlying concepts adopted in the GKS specification reflect experience gained from many of the earlier graphics systems mentioned in chapter 1, although some of the features of GKS are new to graphics programming. The production of standard language bindings for GKS has also received considerable attention. This work has involved mapping the GKS abstract data types and function names onto those types and naming conventions of the host programming language. Currently, bindings for Fortran77 and Ada have been accepted by the British Standards organisation.

3.2 The Graphical Kernel System(GKS).

The Graphical Kernel System(GKS) is the definition of an Applications Interface(API) for computer graphics. Its definition is independent of any programming language or operating system. GKS represents the combination of work in all classes of computer graphics, simply because, as a standard that is to survive, it must encompass all current practices and allow the flexibility to respond effectively to new graphics devices or techniques. Experts from many countries have contributed to its specification. This makes GKS an ideal vehicle to introduce the main features and concepts underlying the current practices in computer graphics. The classes of primitives, picture segmentation, input models, attributes, and device independency are all key issues in todays graphics community. For GKS to survive its underlying structure must enable it to adapt to the future needs of graphics systems.

GKS was designed to address as wide a range of two dimensional applications as possible, meaning that a full specification implementation of GKS may be overly comprehensive and consequently too large for certain simple

applications or small microcomputer configurations with their limited main memory or secondary storage space. Under these circumstances it would be advantageous to supply only a subset of the GKS functions. In order to prevent an arbitrary selection of these functions being chosen, which would defeat the object of creating a standard, a well defined functional split has been defined as a part of the standard. Two axes of functionality are defined which specify graphical output and graphical input respectively. For output there are three levels of increasing complexity ranging from "0" to "2". A higher level of functionality is always a superset of the level below it, which avoids inconsistencies. For input there are again three levels, this time ranging from level "a" to level "c" which also corresponds to an increase in functionality. Hence, a three by three matrix is formed, giving 9 well defined levels ranging from the simplest at level "0a" up to the full specification GKS at level "2c". For graphical output, level "0" provides functions suitable for viewing applications. Level "1" includes functions for picture segmentation and segment manipulation which are more applicable to interactive applications. At level "2", functions for run time picture storage and retrieval are defined. On the input axes, level "a" provides no input, level "b" provides input that is synchronized to the operators actions, that is, the application program must halt until the input from the operator has been satisfied. Level "c" provides asynchronous input where input processing may occur concurrently with the application program.

3.2.1 The GKS data structures.

The GKS standard introduces several abstract data types which are used to define the composition of its internal data structures and the parameters which will be passed from the application program to GKS and vice versa. The GKS

abstract data types may be readily modelled by today's high level programming languages. The working environment for GKS is initialised from a class of internal data structures called description lists. The GKS description list contains such information as the maximum number of workstations and the maximum number of simultaneously open segments that may be supported by a particular implementation of GKS. A description list is also required by GKS for every different workstation type that is to be driven. Each description list serves to define the capabilities of the physical devices comprising the workstation.

At any time during the execution of an application program, GKS will be in just one of five well defined and different operating states. For each of these operating states, only a subset of the complete range of GKS functions is available. There are data structures that maintain a record of the current state of GKS and its associated workstations. The availability of these state structures is dependent upon the internal state of GKS and the trail of commands that the application program has previously performed.

3.2.2 The GKS Workstation Concept.

A GKS workstation is a collection of abstract graphics input and output devices that will map onto a collection of physical hardware devices. A workstation consists of zero or one logical display surface and zero, one or more logical input devices.

The concept of an abstract graphics device called a workstation is fundamental to the definition of GKS and helps achieve device independence by presenting the application programmer with a well defined workstation interface rather than the underlying physical hardware. The term

"workstation" is not to be confused with the popular use of the word, which is usually associated with a turnkey system for computer aided design.

Workstations can exist in any one of the states, open, closed, or active. Opening a workstation establishes a connection between the workstation and the application program. Putting a workstation into an active state will cause GKS to route all graphical output to it, including the data to be stored by the workstation as a segmented display file. The input devices of a workstation may only be used when it is in the active state. An application may have as many open and active workstations as it requires. When a workstation is no longer required by the application program it must be deactivated and closed, at which point its connection to the application is severed, thus releasing it for future use by either the same, or another application program. A function is provided to clear the workstation display surface and purge its local picture segment store.

There are six categories into which a workstation may be placed. The first three of these categories are OUTPUT, INPUT and OUTIN and refer to workstations that have underlying physical hardware that will be used to provide the man machine interface. The OUTIN workstation combines both output and input devices, enabling interactive applications to run on a single workstation. The GKS specification includes three other workstation categories that are responsible for both the temporary and long term storage and retrieval of graphical information. These are known as the Workstation Independent Segment Storage(WISS), the Metafile Output(MO) workstation and the Metafile Input(MI) workstation. These workstations receive their information from the same point in the viewing pipeline as the former three workstation categories and have been defined as workstations for the purposes of control. Hence these

workstations may be opened, activated and closed as normal. The WISS allows temporary storage of graphical data which may be reused later by being routed to other active workstations. For permanent data storage in GKS, the MO workstation must be used. Information is stored on the MO workstation in the sense of an audit trail which may be interpreted at some later time on the same, or on a remote installation using an MI workstation.

When drawing primitives it is often necessary that the application must distinguish between them in some way, for example, when the lines representing two different sets of data on a graph must appear unique. The GKS workstation introduces a mechanism whereby an application program may specify that primitives be displayed in such a way that an operator can distinguish between them. The mechanism is device independent and can be tailored to best suit the capabilities of the physical output device of the workstation. For example, two lines may be distinguished by colour on a colour device, or by line type on a monochrome display device.

Ideally the state of the display surface should represent the current state of GKS at all times. For example if an application program has requested the generation of primitives, they should all appear on the screen as soon as possible after the request to draw them has been made. In certain circumstances it may be more efficient to simply buffer up commands to a workstation and then send the completed buffer all in one operation. For example, remote devices connected via serial links require buffering to improve efficiency of data transfer. The GKS specification realizes this and provides a mechanism to support buffering in a controlled way. Each workstation may be given a defferal state which specifies when the command buffer should be flushed. There are several options ranging from "As Soon As Possible"(ASAP) to "Before

the Next Interaction Globally"(BNIG) through to "At Some Time"(AST). If output is delayed, then the state of the display surface will be undefined for certain periods of time.

3.2.3 Coordinate Systems.

Since GKS is a two dimensional standard, its coordinate systems are defined in 2D space, there being a total of three systems defined. The first is known as world coordinate space and is the application programmer's coordinate system. Points defined in World Coordinates(WC) are mapped onto the second coordinate system which is a virtual space measured in Normalised Device Coordinates(NDC), these being constrained to lie in the range 0 to 1. A normalization transform maps WC to some part of the NDC space, enabling the applications programmer to effect picture composition upon NDC space. Several normalization transforms may be defined simultaneously to map different WC systems onto NDC space. Normalization transforms are not constrained to maintain the same aspect ratio as the WC system. The normalization transform maps a window onto the complete 2D space and is defined by a rectangle with its edges parallel to the coordinate axes. The viewport is what the window is mapped onto and is defined to lie within the bounds of NDC space. The NDC space forms an abstract viewing surface, which represents the workstation display surface.

A further transformation, called the workstation transformation is defined to transform NDC space onto some part of the physical display surface which is measured in Device Coordinates(DC). Only one workstation transform may be defined at any time and it must maintain the aspect ratio of the workstation window whilst mapping as much of this window as possible onto the device viewport. Figure 3.1 shows how the mappings take place, and how they

correspond to multiple workstations. Each normalization transform has a priority associated with it to define which primitives should obscure others when overlapping normalization viewports have been defined.

3.2.4 Graphical Output, Primitives and their Attributes.

Graphical output is built from a small set of display primitives that address the functions of both vector and raster systems. The GKS primitives are abstractions of the basic operations that are commonly performed by graphics equipment. For example, the GKS primitive Polyline will map onto a device's line drawing function. The complete set of GKS primitive types are, Polyline, Polymarker, Text, Fill Area, Cell Array and Generalized Drawing Primitive. Each primitive type has associated with it a group of attributes that control various aspects of that primitive. There are three types of attribute, called geometric, non geometric and identification. The first two affect the appearance of the primitive, while the third attribute type is used in connection with input.

Geometric attributes control the shape or size of a primitive, and are independent of the type of graphics device in use. Where applicable, geometric attributes are specified in the user's world coordinate system and are subject to the same transformations as primitives. The current values of all the geometric attributes are stored in the GKS state list and are bound to the primitives after the WC to NDC transformation has been performed. A range of GKS functions are provided to enable the geometric attributes to be updated dynamically by an application program.

Non geometric attributes control all other aspects affecting the appearance of a primitive, such as line type for polyline, and marker size scale factor for

polymarker. Non geometric attributes may be defined in two different ways, resulting in a workstation independent or a workstation dependent representation of their effect. The selection between one of these two representations is made using a further attribute called an Aspect Source Flag(ASF). For each non geometric attribute there is a corresponding ASF which may take on one of the two enumerated values of "bundled" or "individual". When the ASF corresponding to a primitive attribute is set to "individual", the workstation independent representation of it is used. For individual representation, the mechanism is the same as for geometric attributes, where the values are stored in the GKS state list and are bound to the primitives before they are distributed to workstations. When an ASF is set to "bundled", its associated primitive attribute is accessed from a data structure in the workstation state list. This data structure consists of several vectors or bundles of attributes which are initially set from the workstation description list, but which may later be reset using GKS function calls. The mechanism for selecting bundled attributes involves the specification of another attribute called the primitive index. One primitive index is defined for each primitive and its value is used to select which of the workstation attribute bundles should be used to define the actual primitive attributes. The indexes for each primitive are stored in the GKS state list and are bound to each primitive before it is distributed to the active workstations. The use of bundled attributes helps to ensure portability of applications since the workstation bundles may be chosen to ensure that primitives with different indices will appear differently when displayed on the workstation display surface. For example, an application program could be developed on a workstation with colour capability, such that different attribute bundles contained different colours to distinguish between the various parts of the display. If such an application is moved to a

monochrome workstation, the workstation bundles may be defined such that linetype, markertype or fill area representation is used to distinguish between the different primitive index representations. Figure 3.2 shows the data flow path of attribute binding to primitives.

The Polyline primitive is defined by the end-points of its constituent vectors. All of its attributes are non geometric; they are linetype, linewidth scale factor and polyline colour index. For vector graphic systems the polyline is the most fundamental primitive that may be generated, but for raster scan systems, the vectors comprising a polyline must be generated by a scan conversion algorithm such as the Bresham algorithm described in chapter 1.

The polymarker primitive generates a series of symbols, whose centre positions are defined by a series of coordinates. Its attributes are marker type, marker size scale factor and polymarker colour index. Predefined marker types include the point, the cross and the circle. On vector systems, the markers must be constructed from vector strokes, but raster systems will probably use a bit mapped character generator to produce the markers.

The text primitive draws a string of characters, starting at a predefined position. Text attributes form a complex set, allowing a very flexible representation to be obtained. There are four geometric attributes and four non geometric attributes. The geometric attributes are character height, character up vector, text path and text alignment while the non geometric attributes are text font and precision, character expansion factor, character spacing and text colour index. The text font and precision attribute caters for simple raster scan terminals by defining string precision text which must merely conform to an initial plotting position. Text of this precision can therefore be produced by bit mapped character generators which are often limited to generating a fixed

orientation, fixed bit pattern representation of characters.

The fill area primitive will shade a bounded area defined by a sequence of vertices that are joined by straight lines. It has the two geometric attributes of pattern size and pattern reference point. Its three non geometric attributes are fill area interior style, fill area style index and fill area colour index. Vector systems cannot efficiently produce solid shaded areas, but the fill area interior style will allow hatching to be produced. Raster scan systems are best suited to producing solid areas, and algorithms like the solid area scan convertor described in chapter 1 may be used.

Cell array is a raster primitive which was included primarily to enable the pixel arrays of a graphics system to be initialised by some external data matrix such as the matrix generated by a digitized camera image. On vector scanning devices it is sufficient to simply display the cell boundaries of the cell array. Although a cell array is best suited to specification in terms of device specific pixel dimensions, GKS requires that it be specified in world coordinates, leaving the responsibility for accurate mapping to device coordinates up to the application programmer. Cell array has no attributes other than pick identifier. The array of colour indices used to define a cell array are pointers into the workstation colour table, and hence are effectively attributes themselves.

The Generalized Drawing Primitive(GDP) was added to allow an implementation of GKS to address the special geometrical output capabilities of a device, such as circle or ellipse drawing. There are no explicit attributes to control the appearance of a GDP, instead, the most suitable and appropriate subset must be chosen from those of the other primitives. For example a circle drawing function would probably use the attributes of the polyline function, but if filled circles were available then these would take the attributes of Fill

Area.

3.2.5 GKS Picture Segmentation.

The set of GKS primitives and their corresponding attributes provide adequate power and flexibility for application programs requiring passive graphical output such as general purpose graph plotting and viewing programs. However, when dynamic picture manipulation is required, the user would be forced to implement his own functions for animation. By dealing from the user level with every primitive comprising the animated object, and perhaps the background as well, primitives must traverse the whole of the viewing pipeline and the efficiency of the system may suffer.

GKS introduces a powerful feature called segmentation which allows the grouping of primitives into a single named structure for the purpose of manipulation as a whole. The primitives to be stored in a segment are bracketed by the GKS calls to "Create Segment" and "Close Segment" such that all subsequent primitives generated after the Create, but before the corresponding Close Segment are both displayed and stored on all the currently active workstations. The segment names, which are supplied by the user during each invokation of Create Segment are unique across all of the segment storage. Primitives are transformed from the user world coordinate space into normalized device coordinates before being transmitted to each active workstation for storage. The clipping indicator and clip rectangle are also stored in the segment along with any attribute information that may be required when manipulating segments. This type of storage is known as workstation dependent segment storage(WDSS) and all segment manipulation must conceptually be performed by the workstation itself. For simplicity, the restrictions on segmentation prevents the nesting of segments since only one

segment may be open on a workstation at any time. Also, once a segment has been closed, no further information may be added, modified or extracted from it for use elsewhere.

Segments may be deleted from all workstations simultaniously or they may be selectively erased. All the segments stored on a workstation are lost after an invokation to clear the workstation is made. A GKS function is also defined to enable segments to be renamed.

Picture manipulation using segmentation is effected through the definition of segment attributes. Each segment has a set of attributes which are unique across all segment storage. The attributes defined are, segmentation transformation, visibility, highlighting, priority and detectability. The transformation attribute allows translation, scaling and rotation to be performed through the specification of a 2x3 matrix consisting of a 2x2 scaling and rotation component and a 2x1 translation component. Two utility functions are defined to calculate the transformation matrix from the more conventional specifications used to define transformations. The segment transformation maps NDC onto NDC and occurs before clipping is performed. When the transformation attribute is changed, its effect is recalculated and the display surface updated, but the segment storage itself is never modified. Hence, the original appearance of the segment may be gained by applying the identity transformation matrix to it. The clipping rectangle is never affected by the segment transformation.

The visibility attribute toggles the segment state between visible and invisible and, in the case of WDSS storage, may exploit the hardware features of the workstation which would otherwise be impossible to address directly from the application programmers level without introducing device dependency. Similarly, the highlighting attribute can toggle highlighting on or off, but the

exact mechanism for achieving this is again implementation dependent. The priority attribute enables overlapping segments to be displayed in the correct order and also defines which segments will be picked on input. Finally, the detectability attribute may be used to mask a segment, making it unavailable for picking during input.

A mechanism for device independent storage and retrieval of primitive data during run time is also defined for GKS. The primitives to be stored are conceptually removed from the viewing pipeline at the point where they are exported to all the active workstations. The management of such a storage mechanism is therefore best achieved by treating it as a workstation and using all the conventional workstation control functions. This storage is known as a Workstation Independent Segment Storage(WISS), and only one may be present in any GKS implementation. Three functions are defined for accessing data from the WISS. The first is a copy function which sends the data stored in the WISS to a named workstation just as if it had been generated from the application level. During a copy, there must be no open segment, hence data may not be transferred from the WISS into some other segment. To move segments onto other workstations, an associate function is supplied, which will move a named segment to a named workstation where it may be used as any other private WDSS segment. Again, no segment may be open during an associate function. A compound segment may be created by using the insert function which will transmit data from the WISS such that it will be inserted into a currently open segment in the WDSS. A data flow diagram showing the association between WDSS and WISS is given in figure 3.3.

Segmentation provides a mechanism to support dynamic picture manipulation which is device independent, but, since the segment storage is performed by the

workstation, its particular hardware features may be exploited to the full. In particular, certain intelligent terminals may be capable of supporting their WDSS remotely from the host GKS machine.

3.2.6 The GKS Input Device Model.

In the same way that GKS provides a shield to the user from graphical output hardware peculiarities by introducing the virtual device output functions of a workstation, it also defines a logical input device model that can be simulated by software to use a wide variety of physical input devices. There are six logical input classes defined by GKS, three of which return truly graphical information to the application. The other three classes are provided to allow an application program to obtain all its input from calls to GKS rather than from the input facilities of the programming language or host computer's operating system. The graphical device input classes are LOCATOR, STROKE and PICK, and these are complemented by the STRING, VALUATOR and CHOICE input classes.

The LOCATOR device class will return a single coordinate to the application together with the normalization transformation defining the viewport in which the returned coordinate lies. The viewport priority is used to resolve problems when a point is chosen that lies in more that one overlapping viewport. STROKE input returns a sequence of coordinates in a manner similar to locator. Overlapping viewports or transitions from one to another viewport are resolved by returning the normalization transform of the viewport in which all the points of the STROKE lie. Hence, during the course of a STROKE input, the normalization value of the STROKE measure process may change. PICK input is used to identify a particular segment. The second naming mechanism, the pick identifier, is used to mark primitives within a segment in cases where

segments contain many individually pickable primitives. The pick identifier allows the application to group primitives into a segment in a most logical manner thus avoiding the generation of many individual segments just to improve efficiency during picking.

The other three device classes return values whose types are most commonly found in typical programming languages. The GKS STRING device class is used for character based input. The VALUATOR device class returns a single real value and the CHOICE input device class returns a single integer value. The methods used to interactively obtain input from devices in these classes is more appropriate to good man machine interface techniques than can be obtained using the input facilities employed in many of today's high level programming languages. For example, the integer value returned by a CHOICE device must represent a single option from a bounded set of options, where the interaction with the operator is supported by a semantically linked echo mechanism. It is also possible for the operator to abort an input request, returning no value to the application, but instead setting an error indicator. This abort facility is also available for the three graphical modes of input.

GKS specifies three different operating modes for the six logical input device classes. The first, called REQUEST mode is the most straight forward and must be provided by a GKS implementation whose functionality complies to level "b". REQUEST mode of input functions in a similar manner to that of a FORTRAN read statement. When a request is made, GKS stops until the request has been satisfied by the operator by indicating either a success, or break action. A second mode, called SAMPLE mode will immediately return the current value associated with the logical input device. Finally, EVENT mode will add input values to a temporally ordered EVENT queue whenever the operator's

actions request so.

To define the precise mechanism of its input devices, and their modes of operation, GKS introduces the logical input device model. Each logical device may map onto one or more physical devices, and one physical device may be used to implement several different logical devices. A GKS input device has six constituent parts, these being its measure, trigger, initial value, prompt/echo type, echo area and data record.

The measure of a logical input device is the result of a measure mapping from the values of one or more physical devices. The measure can be visualised as a particular state of a measure process, where each state corresponds exactly with a logical input value. The measure process is in existence for as long as the interaction is taking place with the logical input device. This implies that in REQUEST mode the measure process of an input device may run when the main GKS process is halted, whereas in SAMPLE and EVENT mode a separate process to the main GKS process must be created when a device enters either of these modes. A multi-process environment must therefore be available to implement SAMPLE and EVENT input.

The trigger of an input device is used by the operator to mark significant moments in time. A logical input device receives a signal from its trigger if it is operating in either REQUEST or EVENT modes. A trigger process exists if there is at least one recipient of its fire signal.

The initial value of a logical input device will be that of its measure process when it comes into existence. The prompt/echo type will be used to provide immediate feedback of the measure value to the operator. The echo area specifies which part of the workstation display surface that may be used for

echoing values. The data record of an input device contains information specific to the class of device to which it is associated. For example the data record of a choice device may contain text strings to indicate the various option meanings to the application program. The format and actual data stored in a data record is therefore GKS implementation dependent, although some mandatory values are specified in the GKS ISO document.

The data and control flow diagrams indicating the relationship between the measure and trigger processes for the three different operating modes is given in figure 3.4.

3.3 Future Extensions to GKS.

GKS is a two dimensional standard, but work is proceeding to define a set of three dimensional extensions to its functionality. The definition of primitives and windowing will include a depth component which will require more complex clipping and hidden surface removal to be performed. For 3D-GKS to achieve a comparable standard of performance to todays 2D-GKS systems, the processing hardware to support it will need to be much more powerful, particularly in handling floating point computation.

A further standard that is emerging from the specification of GKS is the Programmers Hierarchical Graphics System(PHIGS)[74] which contains functions to create and manipulate nested graphical objects. The processing power of systems for PHIGS will be more demanding than that of 3D-GKS[73] and is currently limited to very specialist hardware configurations.

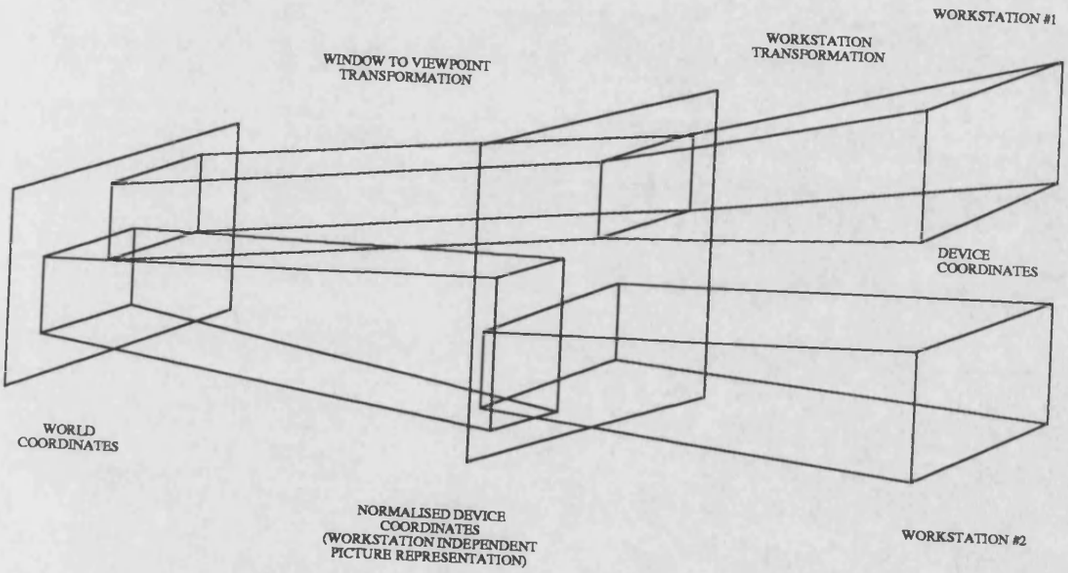


FIGURE 3.1 GKS TRANSFORMATION AND COORDINATES REPRESENTATIONS

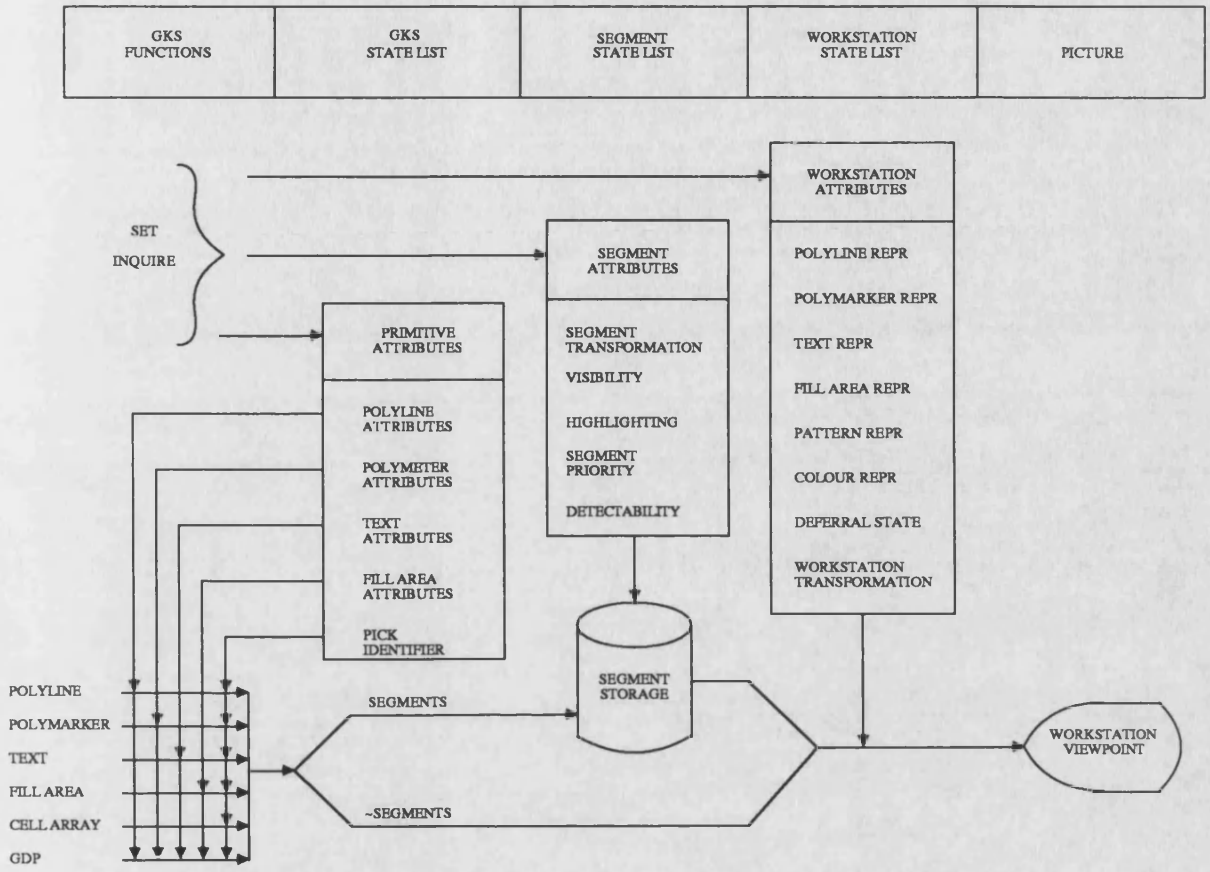


FIGURE 3.2 THE BINDING OF ATTRIBUTES TO PRIMITIVES IN GKS

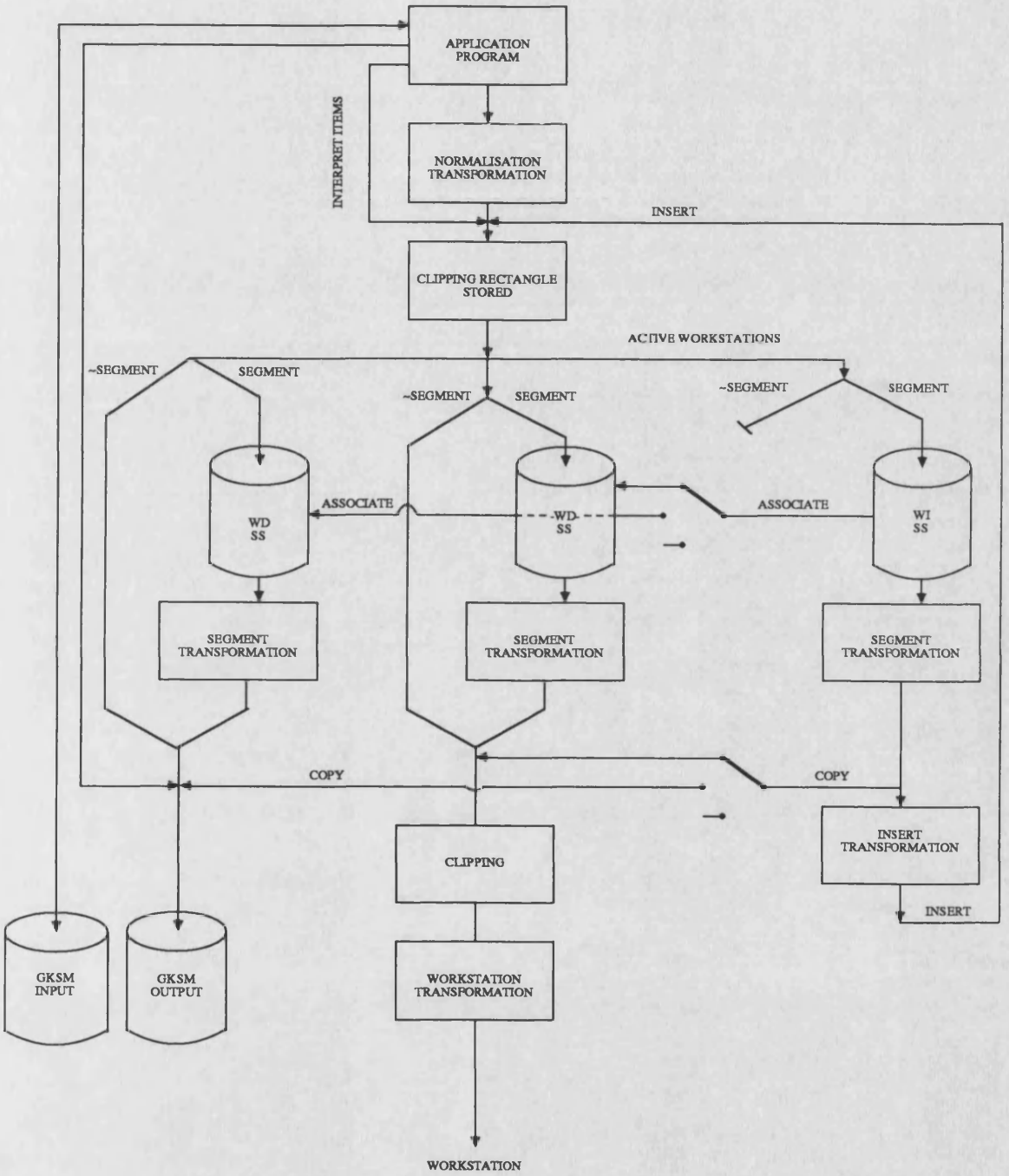
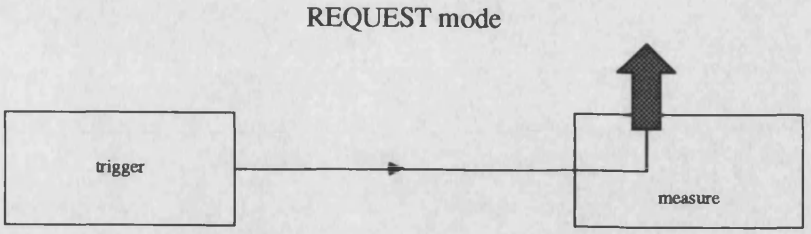
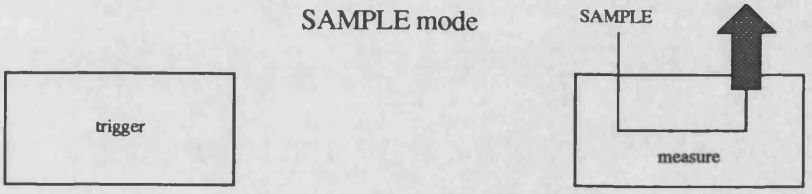


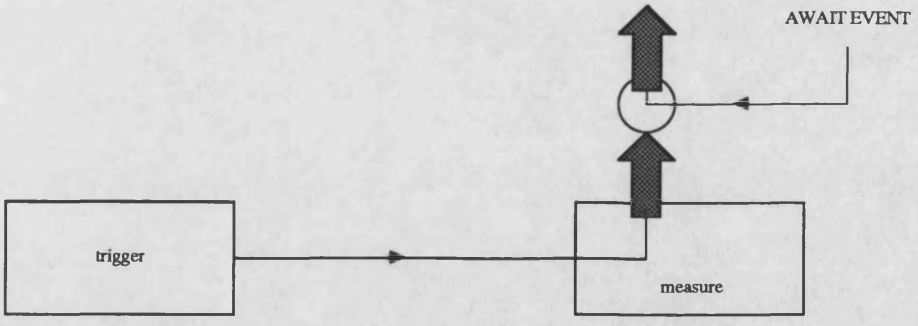
FIGURE 3.3 DATA FLOW CHART FOR GRAPHICAL OUTPUT IN GKS



A single value is returned to the application program on the trigger firing. Interaction lasts for a single request.



A value is returned for each call to SAMPLE. Multiple calls to SAMPLE may be made in a single interaction.



The values and device identification are sent to a single queue on the trigger and removed by a call to AWAIT EVENT

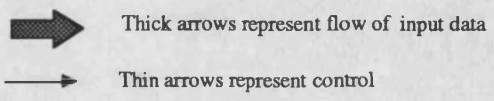


FIGURE 3.4 MEASURE TRIGGER RELATIONSHIPS FOR THE 3 GKS INPUT OPERATING MODES

CHAPTER 4

THE DEVELOPMENT OF TWO RASTER SCAN GRAPHICS SYSTEMS

4.1 Introduction.

This chapter describes the implementation of two raster scan graphics systems for use with the computer hardware that was described in chapter 2. Both designs use single chip CRT controllers to minimize cost and size and to simplify the implementation details. Controllers from NEC and Hitachi have been used, each representing a state of the art graphics device at the time of design.

The graphical functionality and specification of both systems has been tailored to workstation applications where device independent graphics packages such as the Graphical Kernel System are required.

4.2 The NEC uPD7220 CRT Controller.

The uPD7220[16] is a dual bus device residing between the host processor and the display memory, for which it assumes complete responsibility. The uPD7220 appears as two eight bit FIFO registers to the host microprocessor. These FIFO buffers are used for command and parameter passing to help match the speed of the asynchronously running microprocessor and graphics processor, which therefore improves their efficiency. The display interface of the uPD7220 can directly address up to 512K bytes of display memory, with the arbitration between drawing and display cycles being controlled by the uPD7220 under instruction from the host microprocessor. Within the uPD7220, a Digital Differential Analyser(DDA) has been implemented using registers, shifters,

adders and counters. Before figure drawing can commence, five of these internal drawing registers must be loaded with values that have been calculated by the host microprocessor from the geometric properties of the figure that is to be drawn. For example, if a line is defined by its endpoints (x_1,y_1), (x_2,y_2), then dx and dy must be calculated such that :-

$$dx = |x_1 - x_2| \text{ and}$$

$$dy = |y_1 - y_2|$$

and then the following parameters calculated such that, dc = 1, d = 2m - 1, d2 = 2(m - 1), d1 = 2m and dm = don't care. Where dc, d, d1, d2 and dm are the names assigned to each of the five drawing registers and 1 is the largest, and m is the smallest of either dx or dy.

The initial drawing direction must also be calculated, being specified as a three bit number describing the octant into which the first pixel will be drawn. The figure drawing logic is capable of executing algorithms for plotting lines, arcs, rectangles and user defined pattern maps into the display memory.

4.2.1 Limitations of the uPD7220.

Preliminary studies of the uPD7220 CRT controller[38], showed that, although it provided good functionality, the requirement that the host processor calculate and load the drawing registers was a considerable overhead. This situation was compounded when the host processor was also having to implement viewing and simulation algorithms for a device independent graphics standard such as GKS. Figure 4.1 shows the additional stages of processing required for the uPD7220 if the graphical interface is defined to lie at the logical primitive level required by GKS. Some method of providing a more

powerful and easily understood graphics interface was required. A second problem with the uPD7220 was its inability to generate interrupts to signal that its FIFO register is full or that it has completed a command. Instead, the host micro-processor must poll the status register of the uPD7220 to ascertain its current state, which is very unsatisfactory in a multi-tasking environment where other tasks could be run whilst the uPD7220 device was busy.

It was proposed that some form of local intelligence to the uPD7220 be implemented to perform the stages of processing illustrated in figure 4.1 which would bring the logical interface of the NEC graphics system up to a higher, more device independent level. An eight bit microprocessor was used to implement this higher level graphical interface to the system. The functionality of the interface was directly applicable to that required by a GKS workstation, thereby simplifying the workstation code associated with a GKS implementation running on the host machine. For the user not wanting to access the graphics system via GKS calls, the higher level primitives emulated by the dedicated microprocessor offered a more easily used device.

The inclusion of a programmable device between the host processor and the uPD7220 allows the partitioning of graphics tasks to provide the optimum performance, and makes future upgrades or re-partitioning possible. Other devices may also be connected to the dedicated microprocessor bus to offer such functions as a parallel interface for a mouse or tracker ball input device, or digital to analogue converters for joystick type controls. Local memory to the dedicated processor may be used for segment storage or to hold the bit patterns associated with programmable character sets.

4.2.2 An Intelligent CRT Controller based Graphics System.

The design for this system was carried out before the Single Board Computer (SBC) system was available, and hence had to provide a multi-processor interface within the framework of the older multi-board computer systems. The communication medium between the host computer and this Intelligent Graphics Peripheral (IGP) was to be the standard multi-board backplane. Two types of interprocessor communication were considered. The first proposal would have involved the construction of a FIFO register to buffer commands between the two processors in a way similar to that of the uPD7220. The advantage of this method was that only a single address location in the hosts input/output device page would be needed. The second proposal was for a shared memory segment which would reside in both the local and host processors address spaces. The use of shared memory provides a very convenient method for interprocessor communication, but some method of resource management must be provided to prevent data corruption of partly used communication packets. Semaphore flags that are shared between the two processors may be used to signal resource usage provided that some form of arbitration is available to solve the contention that arises when both processors simultaneously access the same flag register.

4.2.3 System Specification.

A general purpose graphics system based on the Thomson-Efcis EF9365 has been designed for the multi-board and single board computers[24]. Its specification was adequate for general purpose viewing with cheap low resolution monitors aimed at the home computer market. For more demanding applications, where shading of surfaces may be required or large amounts of data must be presented, the resolution and colour capability must be increased. The specification of picture resolution for the NEC graphics system was

dependent on several factors relating to the limitations of the discrete devices employed in the design. The uPD7220 features a fully programmable display format and hence it is only necessary to decide upon the maximum pixel clock frequency that will yield the desired resolution whilst complying with the timing specification of the display monitor. The pixel clock frequency was chosen to give a display format suitable for monitors in the medium resolution range described in chapter 1. A frequency of approximately 30MHz was chosen, which gave a 720x720 pixel resolution non-interlaced display or a 1024x720 pixel resolution interlaced display. The timing calculations for these two display formats are given in appendix A.

The need to display realistic shaded images led to the specification of an eight plane system, giving 256 simultaneously displayable colours. Techniques of image plane swapping, known as double buffering, or the use of pixel plane priorities could also be implemented with a multi-plane system, provided that the resulting loss in the number of simultaneously displayable colours could be tolerated. A colour look-up table was specified to ensure that best advantage could be made of this eight pixel plane system. A hybrid colour look-up table was used, which reduced the chip count to a minimum and eased system design. The device chosen contained three four bit Video Digital to Analogue Converters(VDAC) whose outputs complied with the RS343 RGB standard [39]. Three random access memories(RAM), each configured as 256x4 bits, with access times of 25ns were used to map the eight bit logical pixel values into 4 bit values representing the respective red, green and blue intensities of the primary video signals.

4.2.4 An intelligent peripheral controller.

A device has been designed by Motorola specifically for use in Intelligent

Peripheral Control (IPC) applications. Two configurations of the basic device are available, these being either with, or without on-chip 2k byte Read Only Memory (ROM). The IPC device selected to provide the processing power for the IGP was the MC68121[40] which has no on-chip ROM. The actual microprocessor used within the IPC is an 8 bit device that is source and object code compatible with the MC6801[41]. The dual bus architecture of this device provides a dual ported RAM that may be shared between its own internal processor and the host processor. The host processor interface is specifically tailored for interfacing to the MC68000 family of devices. The method of locking resources described above, is implemented by the MC68121 using shared semaphore registers with associated hardware to automatically resolve simultaneous accesses in favour of the IPC. The IPC may be operated in eight different modes, with each mode providing a different pin assignment to alter features such as address space size and the number and type of I/O ports. In this application, the fully expanded mode has been selected to give a 64k address space so that Random Access Memory(RAM), Electrically Programmable Read Only Memory (EPROM), graphic control registers, and the NEC7220 device may all be memory mapped in the local address bus of the IPC. The host bus interface consists of six shared semaphore registers and 128 bytes of dual ported RAM that also appears in the local address space. The bus protocol supports asynchronous transfers by the use of a Data Transfer ACKnowledge(DTACK) signal that is directly compatible with the MC68000 family of devices. The local bus of the MC68121 supports synchronous data data transfers compatible with the MC6800 family of devices.

4.2.5 Hardware Implementation.

Three extended Eurocards were used to house the prototype IGP system, the

first being the controller card which consisted of the IPC and uPD7220 devices and their associated logic. The other two cards were used for the bit mapped memory. The host MC68000 system backplane supplied power to all three cards and carried the control and data signals from the rest of the computer system to the controller card. A local graphics bus linking the controller and the bit mapped memory cards was routed along the front of the system rack using ribbon cable.

A 64k byte addressing range was provided by the IPC when operated in the single expanded mode. The lower 8 address lines and the 8 data lines of the local bus were multiplexed using the IPC Address Strobe(AS) signal. The local bus timing diagram for read and write cycles is shown in figure 4.2. A 2Kbyte EPROM device was mapped into the top of the address space so that permanent service routines could be supplied for the interrupt vectors and reset vector which reside in the very top 16 locations of memory. The EPROM was initially programmed with a machine code executive monitor that provided utilities to aid in firmware development and was later extended to include the debugged graphics applications code as well. Random access memory was also included in the address map. For simplicity, two static devices of 2Kx8 bit organisation were used. More devices could have been added but the physical restriction of on board space prevented memory expansion. The uPD7220 occupied two locations in the address map.

To support the resolution and colour capability of the system, a bit mapped memory, organised as eight planes of 64kx16 bits, was required, giving a total capacity of 1 Mbyte. Dynamic memory devices offered the only sensible method of implementation this array which kept both the cost and size of the frame store within acceptable limits. At the time of design the largest DRAM

devices available were of 64kx1 density. A frame buffer was constructed from 128 64kx1 DRAM devices, which together with control logic and video shift registers and buffers occupied 2 extended Eurocards.

An eight bit colour register was mapped into the IPC address space. It was set before drawing to the frame store commenced so that its data could be directly used to program the pixel colour number during subsequent drawing cycles. This feature made multi-plane drawing cycles as fast as those of a single plane. However it did remove the logical modify operations of the uPD7220 because it prevented it from executing a true read-modify-write cycle during drawing. In fact, during the read part of a read-modify-write cycle, the uPD7220 was forced to read a bit pattern of all zeros, and a drawing mode used to ensure that all the appropriate bits of the current word would be set. Two different techniques could have been used to permit the uPD7220 to do logical operations on data in the frame store. Firstly, multiple invocations of each drawing command could have been given, one for each bit of the logical pixel but, in an application where there are eight bits representing each logical pixel, this technique will degrade performance more than eight fold. The second technique required multiple CRT controllers such that there was a uPD7220 device to control each bit plane. Commands would be issued to all controllers simultaneously and executed in parallel. Hence, no performance overheads would be incurred when using multi-plane frame stores. The quantity and cost of hardware however outweighed any advantages that may be gained by using multiple uPD7220 devices.

The restrictions imposed by using an external colour register did not affect the system performance when using a graphics standard such as GKS. The concept of data storage within the frame store is not supported by GKS, and

commands only allow drawing upon a display surface rather than the processing of data that may already be there. The use of a simple colour register is wholly adequate in these instances.

It became apparent that some method of reading data from the frame store would be advantageous for displaying graphics cursors. A cursor may be displayed by reading data from the store and executing an exclusive-or operation on it. The result is to invert all the bits representing a pixel and hence modify its appearance (colour) on the screen. When the cursor needed to be moved, a second exclusive-or operation was used to restore the original colour number to all of the pixels in question. The display memory could have been dual ported, but problems associated with arbitrating access between the uPD7220 and the IPC, and the level of hardware modifications required to achieve this ruled out dual porting. Instead, the method used involved catching the required pixels as they were being shifted out of the video shift registers. The hardware to implement this feature consisted of a programmable counter, three latches, a tri-state buffer and a small amount of logic. Although this implementation was inefficient, it was powerful enough to display a moving cross hair cursor under control from a joystick input device without noticeable delay.

The uPD7220 provides support for picture zoom using a method of pixel replication by repeatedly accessing a single scan line from one to sixteen times before passing to the next one. Thus the picture may be zoomed in the vertical direction without external hardware support. To provide zooming in the horizontal direction requires fast external devices to the uPD7220 that are capable of operating at the pixel clock speed of 30Mhz. These devices must divide the pixel clock by the zoom factor and condition the loading of the video

shift registers to ensure that a contiguous display is obtained.

Scrolling of the display is achieved by modifying the start address of memory where picture refresh scanning is to commence. This feature is wholly controlled by the uPD7220 and permits smooth vertical scrolling and horizontal scrolling in discrete steps of sixteen pixels.

Upon completion of a graphics request from the host processor, the IPC executes a write cycle to one of its I/O ports to set and then reset a flip-flop register. The output of this register is used to signal that an interrupt should be generated by a backplane interrupt state machine for the attention of the host processor. When the host processor performs an interrupt acknowledge cycle, the interrupt state machine will clear its request, if appropriate. This technique enabled graphics requests to be issued to the IPC in the most efficient manner. A block diagram of the complete IGP system is shown in figure 4.3.

4.2.6 System firmware.

The production of firmware for the system was eased by the use of various software tools. The TRIPOS operating system had an assembler for the MC6801 capable of producing assembled output in "S" record format[42] which was suitable for transmission to a commercial EPROM programmer. A machine code executive monitor for the MC68121 was available from Motorola to enable efficient program development for this microprocessor[43]. This monitor resides in 256 bytes of the 2K byte EPROM and offers functions to load, test and initiate user application firmware in the local bus RAM. The development environment was completed by a program written in BCPL to read the "S" record files produced by the assembler and use these to load and run graphics applications programs by using the functions of the MC68121 monitor. The

program allowed the start address and stack pointer to be set anywhere within the local RAM address space.

The set of functions to be performed by the software on the IGP system were directly modelled on those necessary to implement the GKS set of output primitives. Some other functions were also added which could have been accessed by the escape mechanism of GKS, or could have been used directly by a non GKS application. A protocol was defined to represent how the data would be transferred through the shared memory area of 128 bytes. The mechanism consisted of a single byte to indicate the function required, with any data following this byte being specific to the function being performed. Appendix B gives a complete specification of the data format required in the shared memory for each command.

Due to the limited amount of shared memory, the driver software running on the MC68000 had to split primitive specifications into blocks, sending each block until the complete primitive had been plotted.

The complete list of functions supported comprise polyline, polymarker, character plotting (bit mapped characters), clear screen to background colour, colour palette programming, zoom factor programming, scroll setting, cursor echo positioning, and character/marker type programming.

All these functions are useful in GKS workstation applications where they may reduce the amount of software simulation required in representing a primitive on a raster scan system. Colour paletting enables a colour number to be associated with a particular combination of intensities for the red, green and blue primary drive signals for the display monitor. When initialisation occurs at system reset, only two colour numbers are set up, colour 0 being black and

colour 1 being white.

The cursor positioning command allows echoing on the screen of coordinates obtained from input devices in interactive applications. The firmware keeps a record of the previous cursor location so that an exclusive-or operation on the pixels associated with it may be used to restore the origional colour numbers before the new cursor location is marked.

The character/marker programming function allows the bit map associated with each character to be programmed, hence, a fully programmable character set was implemented. The complete set of bit maps were stored in the local RAM area, from where they were read into the uPD7220 pattern ram before being drawn into the display memory during marker or text plotting. Each character was specified as a bit map on a grid of 8x8 pixels. Character and marker size scaling between 1 and 16 times was programmable, this being automatically achieved by the uPD7220 as it was drawing.

The zooming and scrolling functions caused the IGP to program the internal registers of the uPD7220 according to the parameters passed from the host microprocessor.

4.3 An Overview of the Hitachi HD63484 CRT Controller.

The uPD7220 CRT Controller had no commercial competition from other manufacturers for over three years, but in 1985 several other custom VLSI design houses released CRT Controllers with functionality that surpassed that of the uPD7220. One such device, the HD63484[17], from Hitachi allowed the construction of a smaller, cheaper, yet more powerful graphics system than that of the IGP.

The need for a new single board graphics system was also increased when the Single Board Computer (SBC) became available. With the addition of a powerful graphics system, the SBC could be used to manufacture a minimum configuration workstation for personal use. The Hitachi ACRTC was chosen for this application because of its compatibility with the MC68000 family of devices.

Figure 4.4 shows a block diagram of the internal structure of the HD63484. Many of the features first exhibited by the uPD7220 are retained, such as the use of read and write FIFOs for data communication. The significant improvement over the uPD7220 comes from the use of three microcoded processors to handle the tasks of drawing, display and raster timing control. The host and frame buffer interfaces, also shown in figure 4.4, operate in parallel with these microcoded processors to maximize throughput.

The drawing processor interprets commands and command parameters that have been issued by the host processor and performs the required drawing operations in the frame store. This processor implements several algorithms for generating lines, rectangles, arcs, ellipses and bounded area fills and performs the logical to physical address translation of the primitive coordinates. The functions performed by the drawing processor therefore replace and surpass those of the IGP.

The display processor manages the screen refresh during display cycles by using a high speed address calculation unit to provide the addresses for window generation and partitioning of graphics and character frame buffer areas.

The timing processor generates CRT synchronization signals and other internal signals required by the HD63484. Registers within the device are

loaded with values that are used by the timing processor. This allows the system to be programmed to suit the display monitor or other external constraints.

Although the HD63484 contains over forty user programmable registers, it only occupies two memory locations in the address space of the host processor. An indirect addressing mechanism is implemented, whereby one of these address mapped registers is loaded with the number of the internal register required, which may then be addressed via the other memory mapped register. A programming model of the internal registers of the ACRTC is shown in figure 4.5. Sixteen bit wide FIFOs are used to transfer commands and parameters to the ACRTC. This is particularly important in high resolution systems as it allows a single (x,y) screen coordinate to be transferred in only two bus cycles. A direct memory addressing interface suitable for the Hitachi HD68450 DMA controller operating in single address transfer mode is supported. Programming the ACRTC with commands and parameters, using chained DMA transfers, enables display segmentation to be supported with the minimum processing overheads. A transparent operating mode of the ACRTC also allows DMA transfers directly to and from the frame store.

Unlike the uPD7220, the HD63484 has a comprehensive interrupt management scheme to indicate the state of its drawing processor and the read/write FIFOs. There are eight different conditions that may generate interrupts, each of which may be enabled or disabled by writing to a particular bit in an interrupt mask register.

The frame store interface can directly handle up to 2Mbytes of random access memory, which can be implemented using either static or dynamic RAM, with refresh for the latter occurring automatically during the horizontal

synchronization pulses. The way in which the HD63484 accesses the frame store and controls colour information shows a significant departure from earlier CRT controllers such as the uPD7220. The memory is organised in a linear fashion, comprising an array of 16 bit values, each of which may be configured to represent logical pixels consisting of 16, 8, 4, 2 or 1 bits. This allows each memory access to conceptually read vertically from the frame store to access complete pixels in one operation. Colour information can therefore be directly controlled by one device whilst incurring no overheads, and maintaining the ability to do logical data operations on data already within the frame store.

4.3.1 System Specification

The final specification of a new graphics system based on the HD63484 was influenced by many factors. The system should be capable of a performance that would equal many of the colour graphics workstations that were currently on the market, but design time and cost were major factors to be considered. The physical size of the final system was also important, being limited of approximately 150 sixteen pin dual in line package equivalents.

To compete with state of the art raster scan graphics systems would have required a video specification of 1280x1024 pixels in a non-interlaced scanning format with a vertical refresh rate of between 50 and 60Hz. The pixel clock frequency for this display format would have been approximately 120MHz, depending upon the timing requirements of the display monitor. The high frequency components of a graphics system of this specification would have required ECL type devices, and design experience with their layout and construction rules. Since standard TTL type devices would still be required for the low frequency parts of the system, many interface devices would have been needed to connect the two incompatible device families. The use of ECL was

ruled out for these reasons, and a study made of the new Advanced Schottky(AS) and FAST TTL devices being produced by Fairchild[44] and Texas[45]. It was estimated that AS devices could be used up to approximately 60MHz when providing logic functions for the HD63484 system. With a pixel clock of 60MHz, a display format of 1280x1024 can only be achieved by using an interlaced scanning technique. The calculations qualifying this statement are given in appendix C. The cost of colour monitors was also considered when deciding upon the maximum pixel rate, there being an almost linear relationship between video bandwidth and cost.

A resolution of 1024x1024 was chosen to give a manageable memory array size and to allow easy factoring when using readily available DRAM parts.

Because interlaced scanning was to be used, a colour monitor fitted with a long persistence CRT was specified to prevent adjacent lines of alternate fields with widely differing luminance values from causing an annoying flickering effect. Long persistence CRTs are suitable for viewing applications but, when animated pictures are required, they can produce unwanted smearing effects. The monitor chosen to interface to the HD63484 system was the Mitsubishi C-6920[46]. The timings for the system were therefore programmed at initialization to be compatible with this device.

When considering the colour specification of a system, there are two main criteria to define. The first is the number of simultaniously displayable colours which will in turn dictate the number of bits per pixel required. A choice of 256 colours was made, giving 8 bits per pixel. This was the same as the IGP system, and gave a memory array size that was managable and easily configured to suit the ACRTC. The second criteria defined how many shades of each primary colour could be displayed. A quantization into 256 steps was

chosen so that a smooth grey scale could be produced. Three, eight bit video digital to analogue convertors were therefore used to convert the pixel information into drive signals for the display monitor. The photographs in figure 4.6 were taken directly from the screen of the monitor to show the three smooth primary grey scales that were generated in this way.

A detailed description of the system timings and ACRTC configuration resulting from the decisions described above are given in appendix D.

4.3.2 The hardware implementation.

A prototype HD63484 system was constructed on two extended Eurocards that were joined together to form a board of equal dimensions to the SBC. The board was fitted with connectors so that it could be plugged into the standard backplane to receive power and SBC bus signals. Wirewrapping was used to interconnect devices.

The system clock of 55MHz was generated by a clock oscillator module[47]. The clock was buffered before being used in the high speed sections of the circuit and was also divided by eight to produce the clock signal for the HD63484. The HD63484 is clocked at 6.875 MHz to give a frame buffer access time of 290 nanoseconds(ns), this being chosen as the minimum possible without violating the DRAM timings. Figure 4.7 shows how a 4 bit D-type flip flop was configured to divide the pixel clock by eight, and to generate a four phase clock. The method is particularly useful for producing shift register load pulses with the minimum of additional logic.

Figure 4.8 shows a block diagram of the frame buffer memory. Its size of 1Mbytes is the same as that used in the IGP design, but the requirement that a single card house the new graphics system dictated that a significant

improvement in packing density for the memory array be made. The increased resolution also put very tight restrictions on timing within the array, meaning that very close coupling between the HD63484 and its memory was essential. The memory configuration required devices with a depth of 64k, so the increased density afforded by the new 256k DRAM devices could not be used. Unfortunately 64kx4 devices based on 256k bit DRAM technology had not become readily available. A solution was found in the use of hybrid memory arrays constructed from 64kx1 leadless devices which were soldered onto small ceramic strips which could then be mounted vertically onto the graphics card to give a three dimensional packing of components. The hybrid packages were available in 64kx8 bit configurations with a minimum cycle time of 230 nanoseconds[48]. A closely packed small memory array helped to minimise propagation delays due to inductance and capacitance. Data multiplexing was achieved with fast buffers which were turned on under control from the address decoding. The timing diagrams for memory access cycles performed by the HD63484 are given in figure 4.9. The very wide tolerance of parameters quoted for the HD63484 caused problems when trying to ensure data set-up and hold timings for the bit mapped memory array when pushing cycle times to their minimum. To avoid problems with device variation, the timing signals intended to control accesses to the frame store were taken directly from the on board clock signal being generated for the HD63484. To ensure accurate positioning of the many control signals required by DRAMS, a delay line with taps at 10ns intervals was used. Accurate timing throughout the memory cycle meant that a delay line of 200ns was required, but the only devices available with 10ns accuracy were 100ns maximum total delay time. DRAM devices require a precharge period at the start of each memory cycle, the specification for the devices used in this design being 100ns[48]. This feature enabled the

novel circuit fragment shown in figure 4.10 to be used to give a 200ns accurate delay from a single 100ns delay line. The memory timing diagram generated by this circuit is shown in figure 4.11. Standard AS TTL multiplexer devices were used to ensure that the row and column addresses were presented to the memory array with the minimum delay.

Address decoding for RMW cycles is performed by conditioning the column address strobe (CAS) signals so that they only become active for the required bank of memory within the frame store. Each memory cycle starts with a Row Address Strobe (RAS) signal to every memory bank so that the unselected banks will receive a RAS only refresh cycle. This technique provided the fastest access method but also increased power consumption, as the whole memory array was always active. Display cycles require data to be accessed from all memory banks and hence the CAS signal is made active to all planes during this time. A Programmable Read Only Memory (PROM) was used to decode the HD63484 signals that indicated what address and what type of memory cycle was under way into the conditioned CAS signals for each memory bank.

Eight shift registers, each of 16 bit length, were required to convert the 128 bits of data accessed from the bit map during each display cycle into a high speed 8 bit pixel stream. The pixel clock of 55MHz produced pixels at the rate of one every 18ns, which represents a data rate of 440 Mbits per second. In the IGP design, two eight bit shift registers had been cascaded, with the serial output of one feeding the serial in of the other. This method was not possible with the 8 bit devices available at the time due to the large data set-up time required at their serial input. Instead, 16 bit devices that had recently been introduced by Fairchild using their FAST technology were specified.

The generation of high speed control signals to condition shift register loading would have been straight foward, if display zooming and hardware cursor functions had not been required. The decision to limit the design to an all TTL implementation provided many problems when trying to implement programmable counters capable of operating at the pixel frequency of 55MHz. Fairchild FAST devices were again used to fulfill the speed requirement. Figure 4.12 shows how the zoom counter was implemented. A hardware cross hair cursor also required a programmable counter because the HD63484 was unable to identify a single vertical pixel location. Instead, the HD63484 provided a signal that was active during the word access containing the required pixel, which relied on external hardware to pick the required pixel from that group of 16. The horizontal component of the cross hair may, however, be controlled solely by the HD63484. Four bits from the programmable attribute word of the HD63484 are used to load the cursor counter which was reset at the start of each display cycle. The required pixel was then identified when the carry output pulse of the counter became active. The x and y cursor signals were then combined to force the pixels that were under the cursor to all zeros.

The flexibility of colour generation is greatly increased by the use of a colour look up table. With eight bits representing each pixel on the screen, a total of 256 colour numbers may be generated at any one time. The colour look up table consisted in this instance of three sets of dual ported random access memory, each configured as 256x8 bits. The colour number of each pixel was used to address these three blocks of memory, with the resulting accessed data being latched into three video digital to analogue convertors to produce the red, green and blue primary drive signals for the colour monitor. Since the colour look up table and video D/A convertors were required to work at 55Mhz, ECL type RAMS with access times in the order of 15ns would have been required,

together with logic level convertors to interface them to the TTL pixel signals. Specialized eight bit video D/A convertors would also have been required to produce the final RGB drive signals. To produce such a colour look-up system would have required a large number of discrete components and a large PCB area. Circuit design and layout would also have been a critical consideration. Instead, a hybrid module complying to the required specification was used. It contained all the functional blocks of a colour look up table, including the video D/A convertors, yet occupied only two square inches of board area. A block diagram of the module is shown in figure 4.13.

Blanking the video pixel stream during flyback proved difficult as the HD63484 signal for this purpose became active one memory cycle too late. This feature was included so that smooth scrolling circuitry could be more easily implemented. Since smooth scrolling was not required, external logic was added to generate a blanking signal from the control signals that indicated display cycles were being produced by the HD63484. A Programmable Array Logic (PAL) device was used to implement the logic required for this task, the PALASM[49] source equations describing this device are given in appendix E.

The photograph in figure 4.14 shows the completed HD63484 graphics system. An IC layout diagram for this system is shown in figure 4.15, and the corresponding schematic circuit diagrams are shown in figures 4.16 and 4.17.

4.4 Performance Comparisons.

The Hitachi HD63484 CRT controller permitted the construction of a more powerful and more flexible graphics peripheral than the IGP that could be used with both the single and multi-board 68000 based computing systems. The HD63484 device itself supported much of the functionality that had previously

been provided by the complete IGP system. The table shown in figure 4.18 compares the hardware performance of the two systems, while the table in figure 4.19 compares their functional specifications. These comparisons show that performance has been either equalled or improved on all accounts in the HD63484 system. Using this system as part of a workstation that supports a device independent graphics interface will therefore bring benefits from the decentralized simulation of primitives, the logical x-y addressing of these primitives and the high resolution, high speed drawing that it performs.

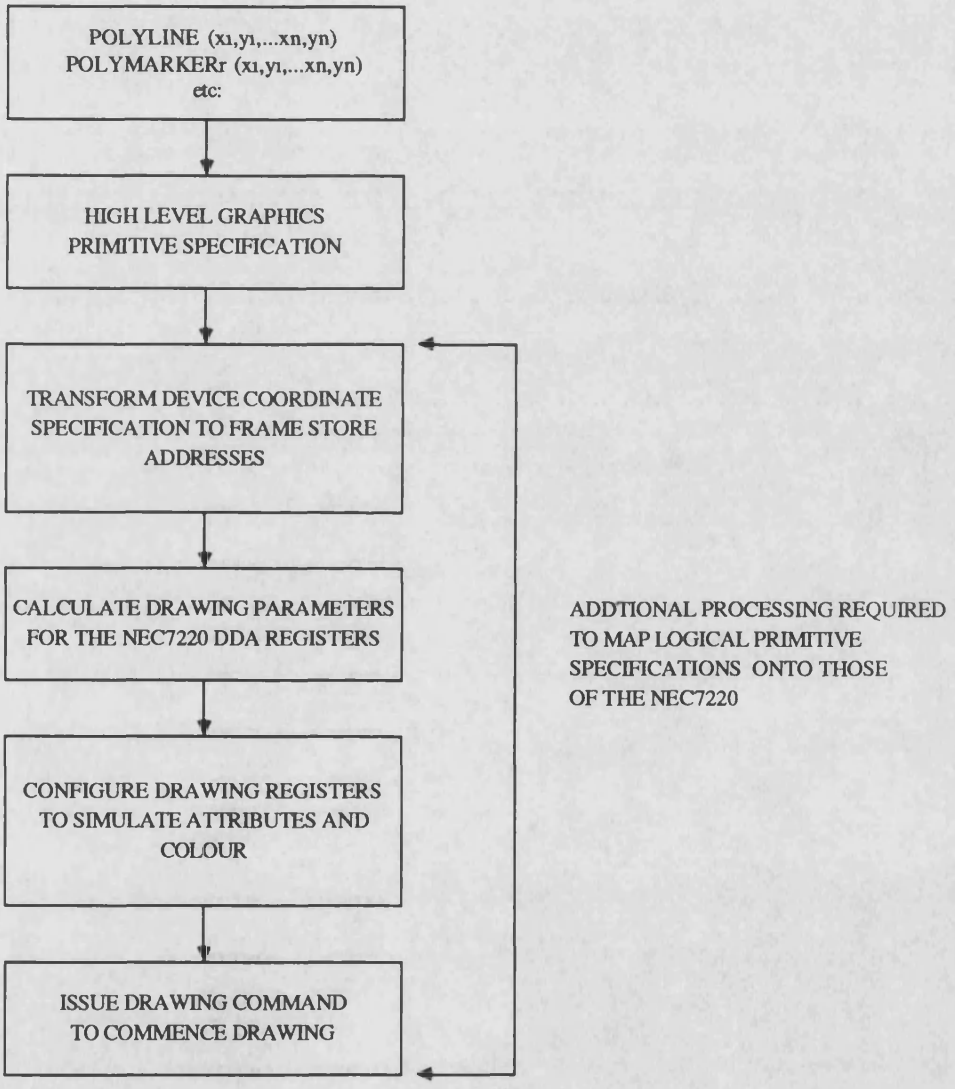


FIGURE 4.1 PROCESSING PIPLEINE FOR THE NEC7220 CRT CONTROLLER

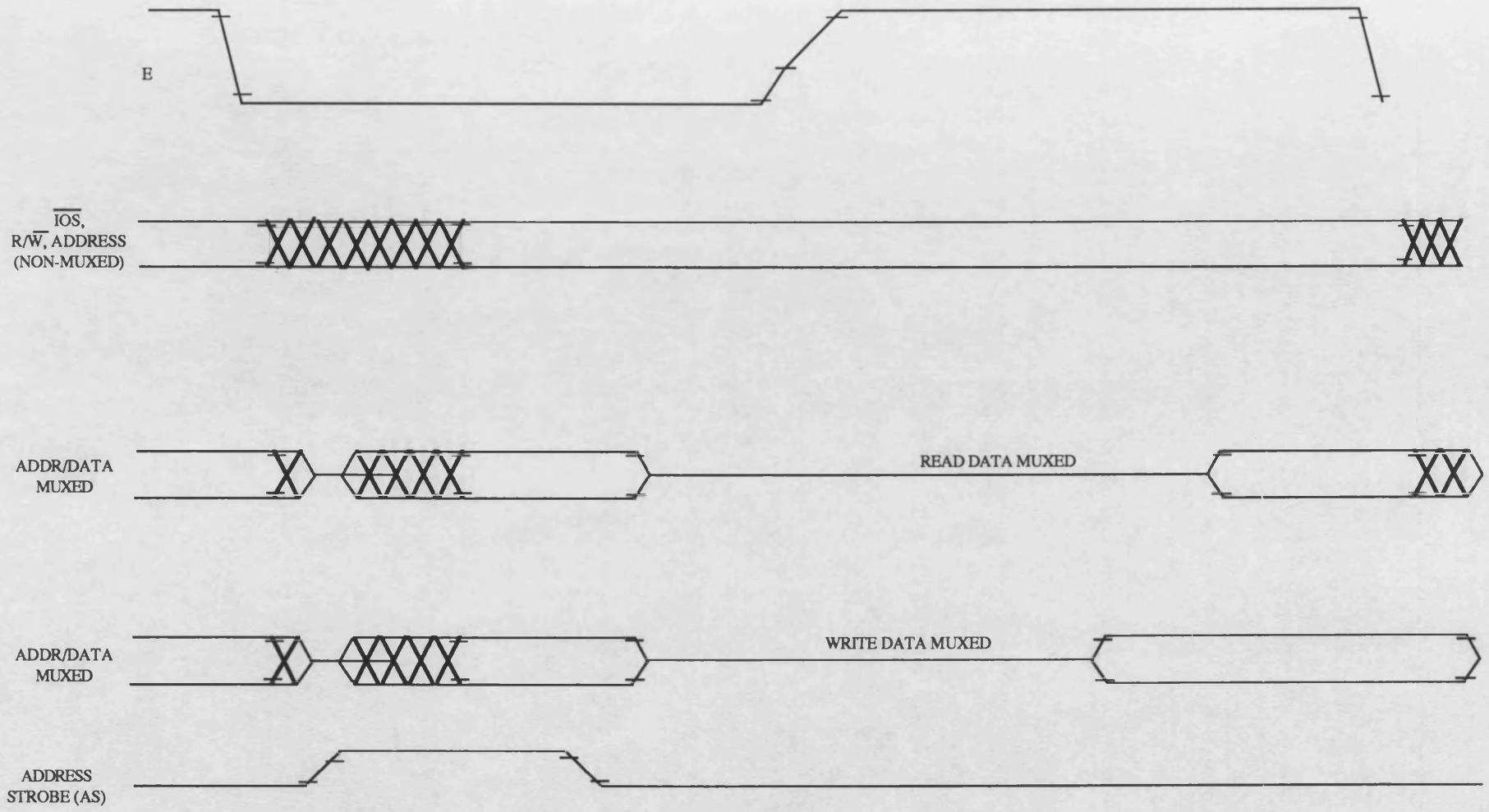


FIGURE 4.2 LOCAL BUS TIMINGS OF THE MC68121 PERIPHERAL CONTROLLER

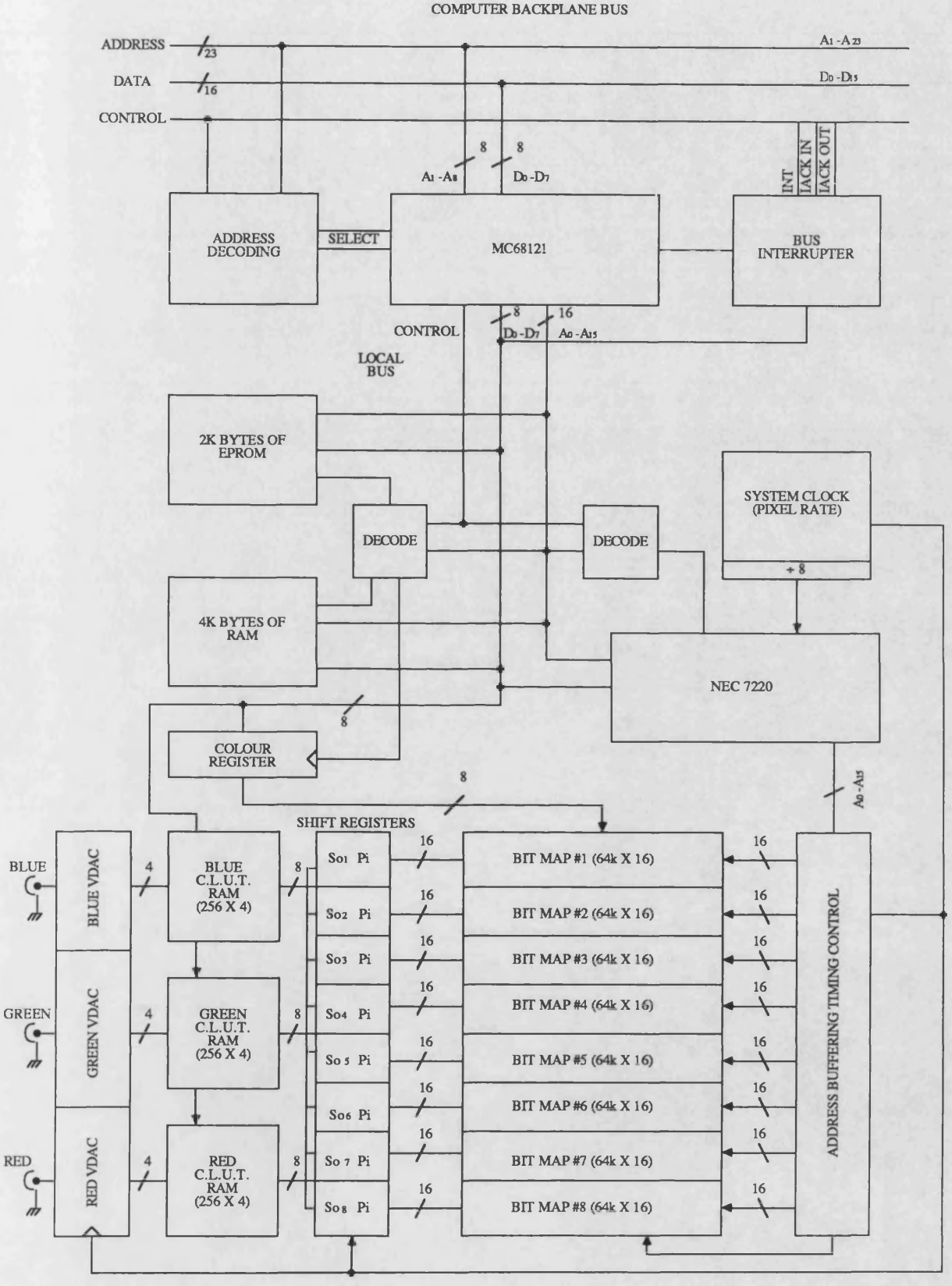


FIGURE 4.3 BLOCK DIAGRAM OF THE IGP SYSTEM.

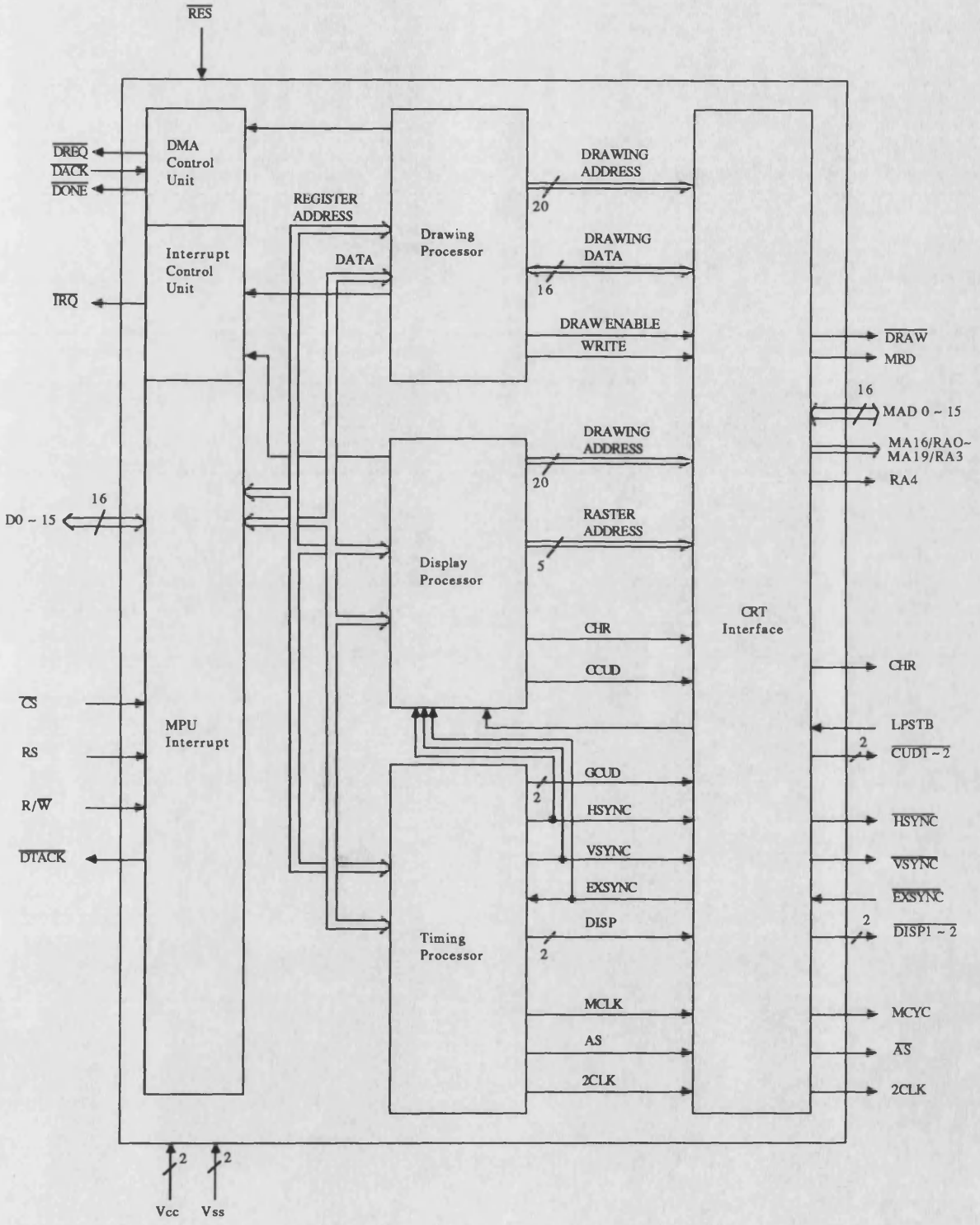


FIGURE 4.4 INTERNAL STRUCTURE OF THE HD63484

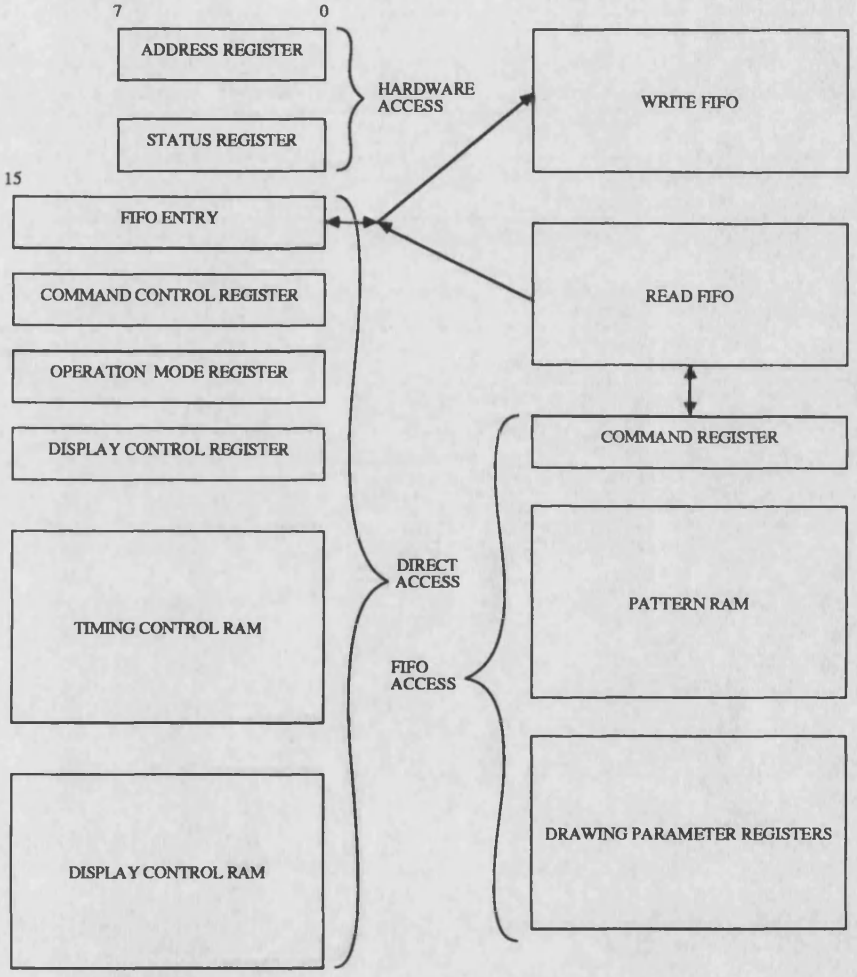
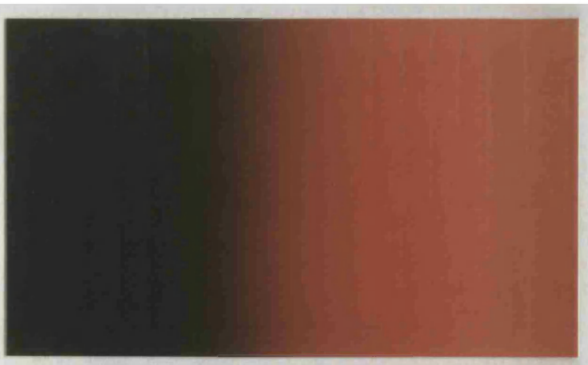
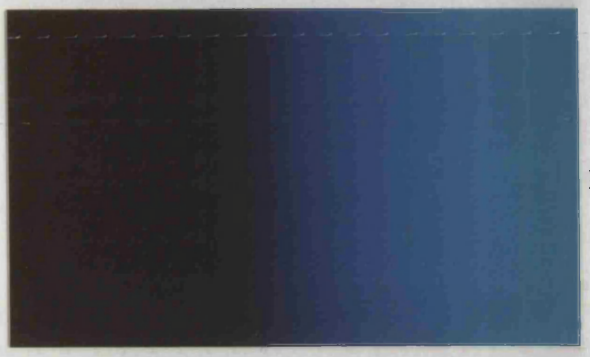


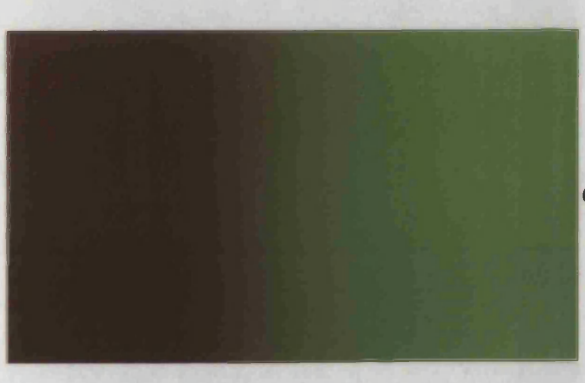
FIGURE 4.5 PROGRAMMING MODEL OF THE HD63484



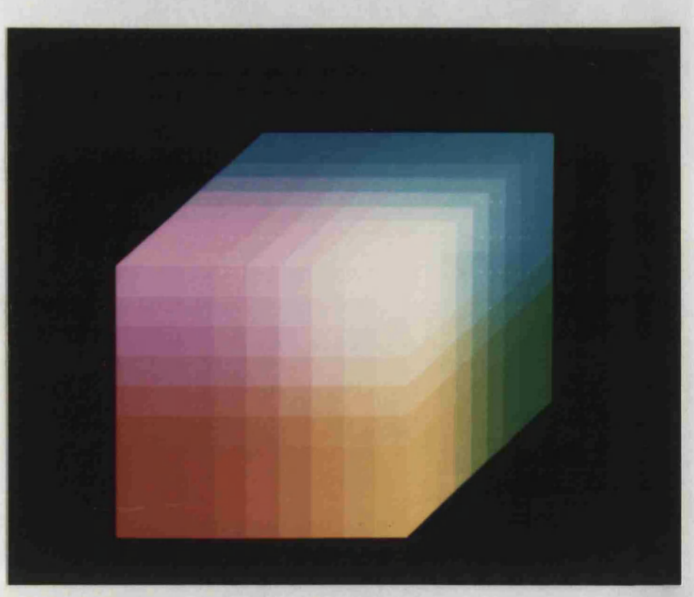
RED GREY SCALE



BLUE GREY SCALE



GREEN GREY SCALE



COLOUR CUBE

FIGURE 4.6 256 LEVEL GREY SCALE FOR THE PRIMARY COLOURS

AND 256 SHADE COLOUR CUBE

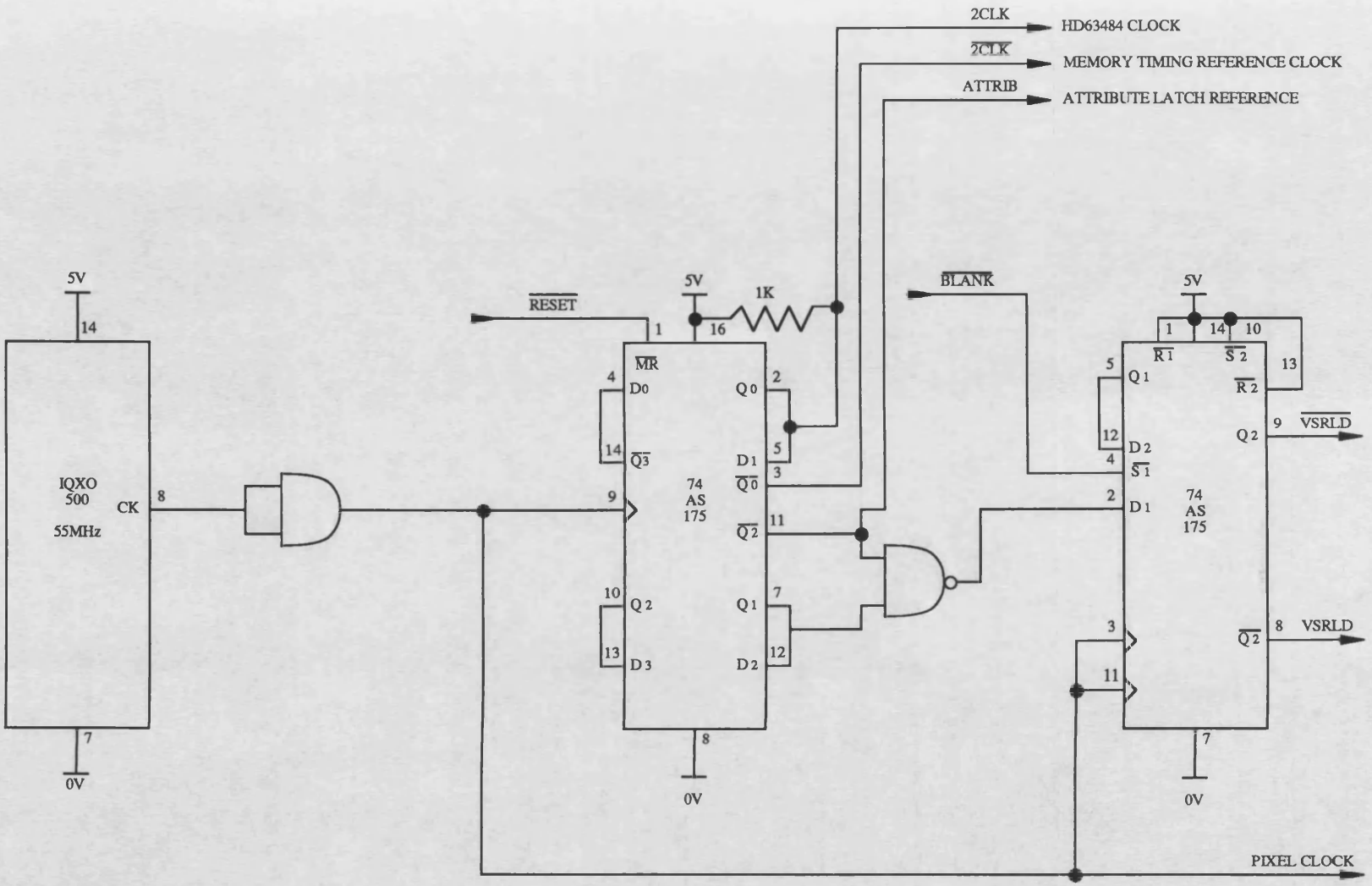


FIGURE 4.7 SYSTEM CLOCK GENERATION

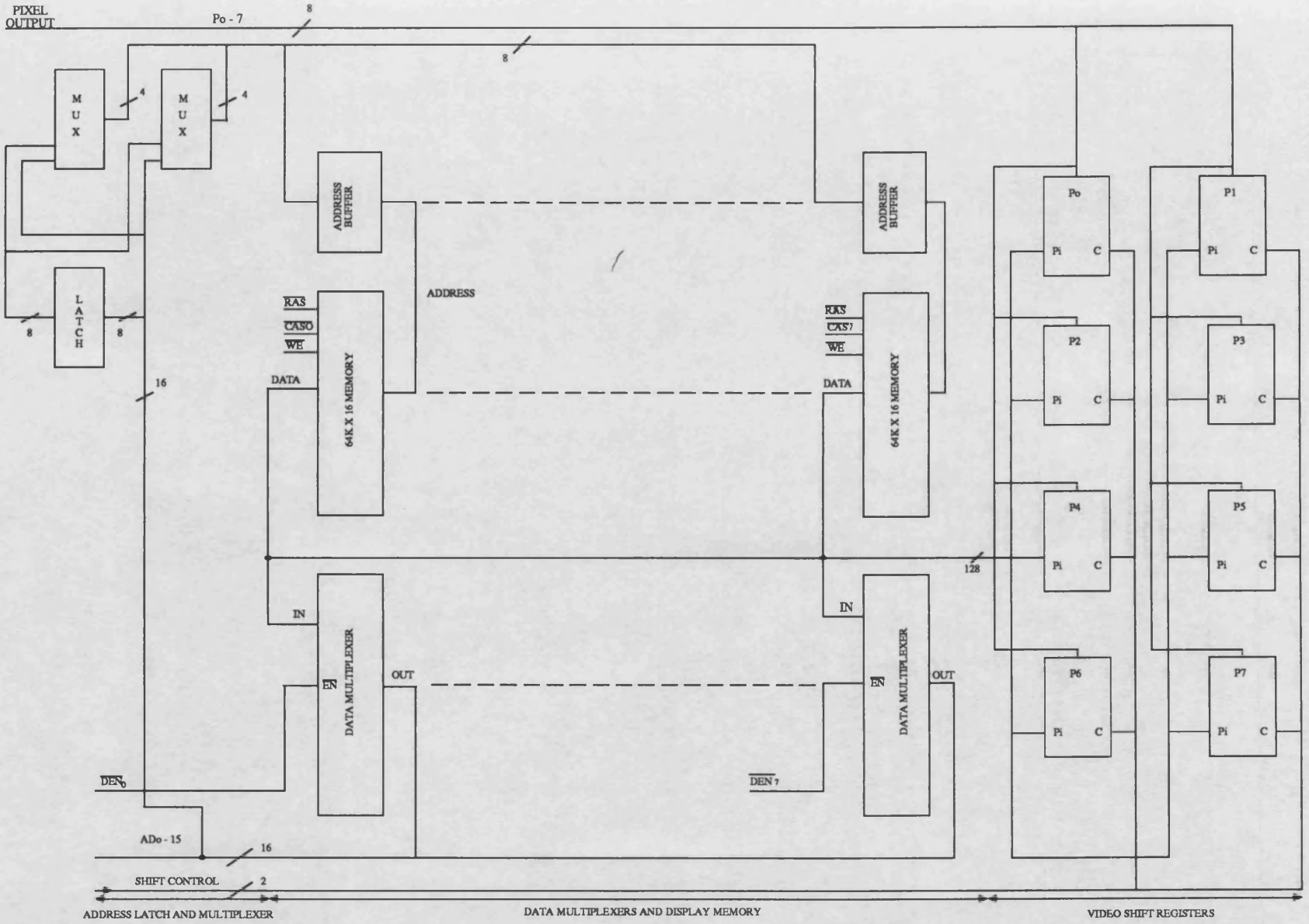


FIGURE 4.8 HD63484 FRAME MEMORY DIAGRAM

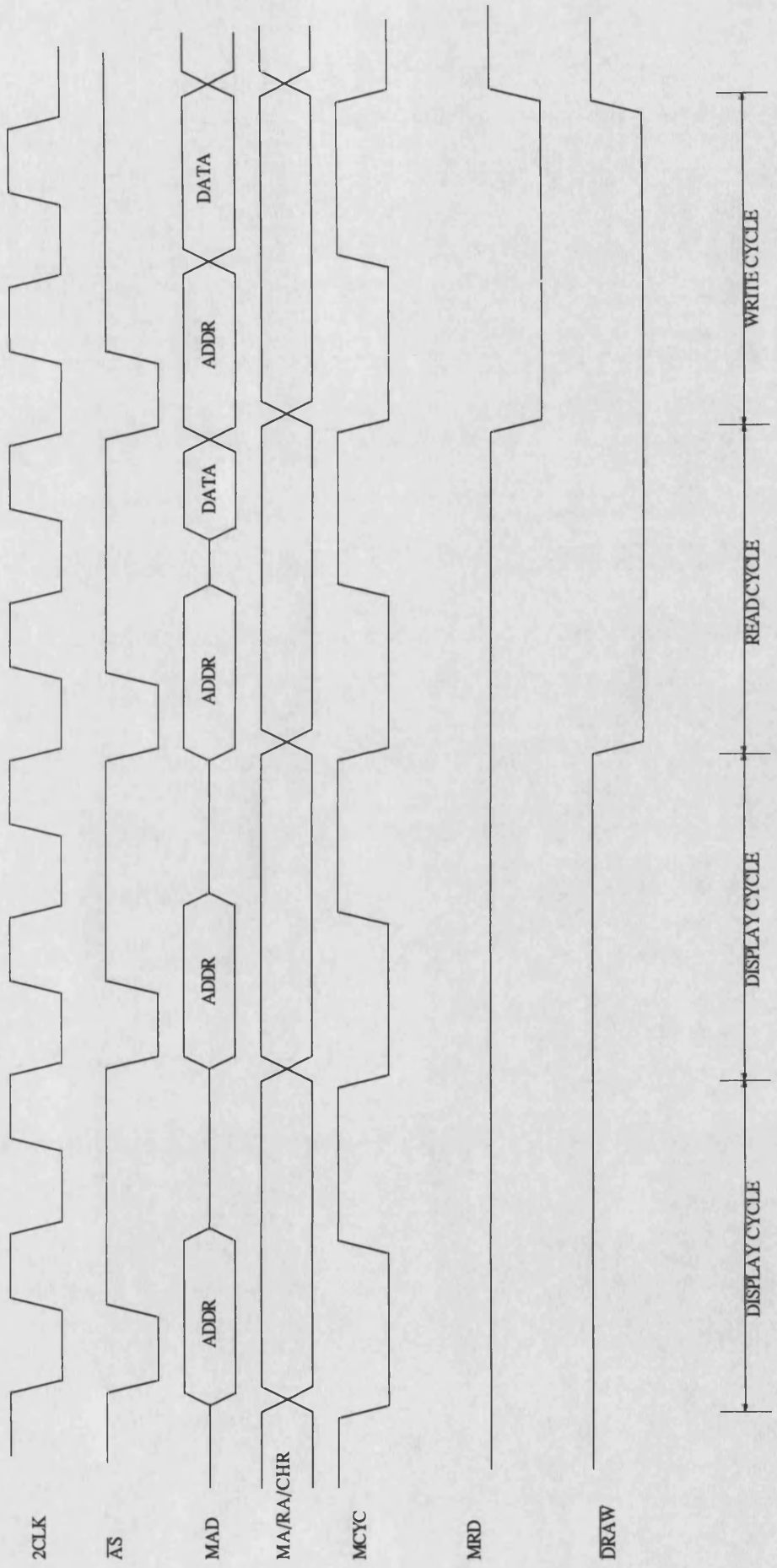


FIGURE 4.9 FRAME MEMORY DISPLAY, READ AND WRITE CYCLE TIMINGS

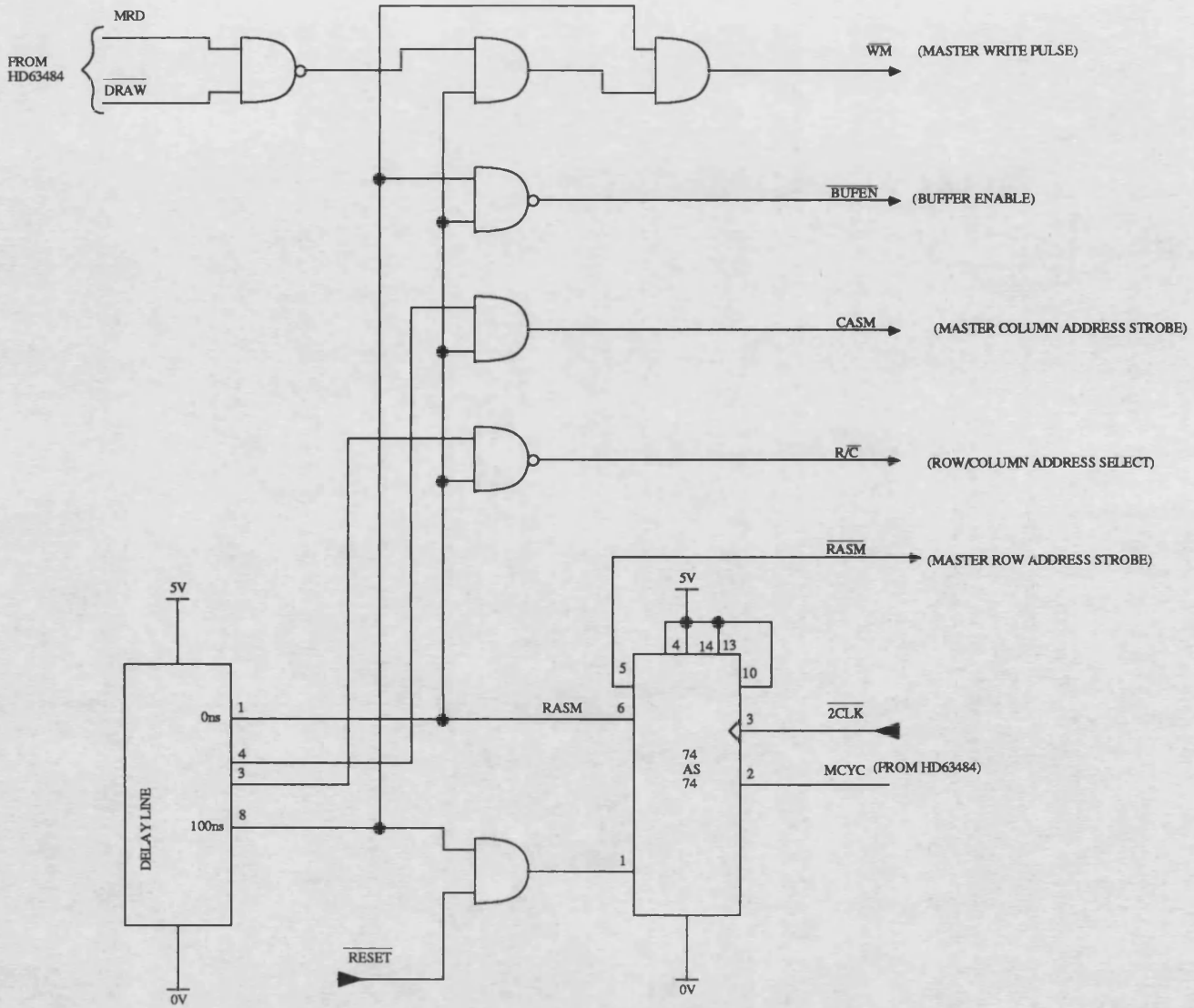


FIGURE 4.10 MEMORY TIMING GENERATOR

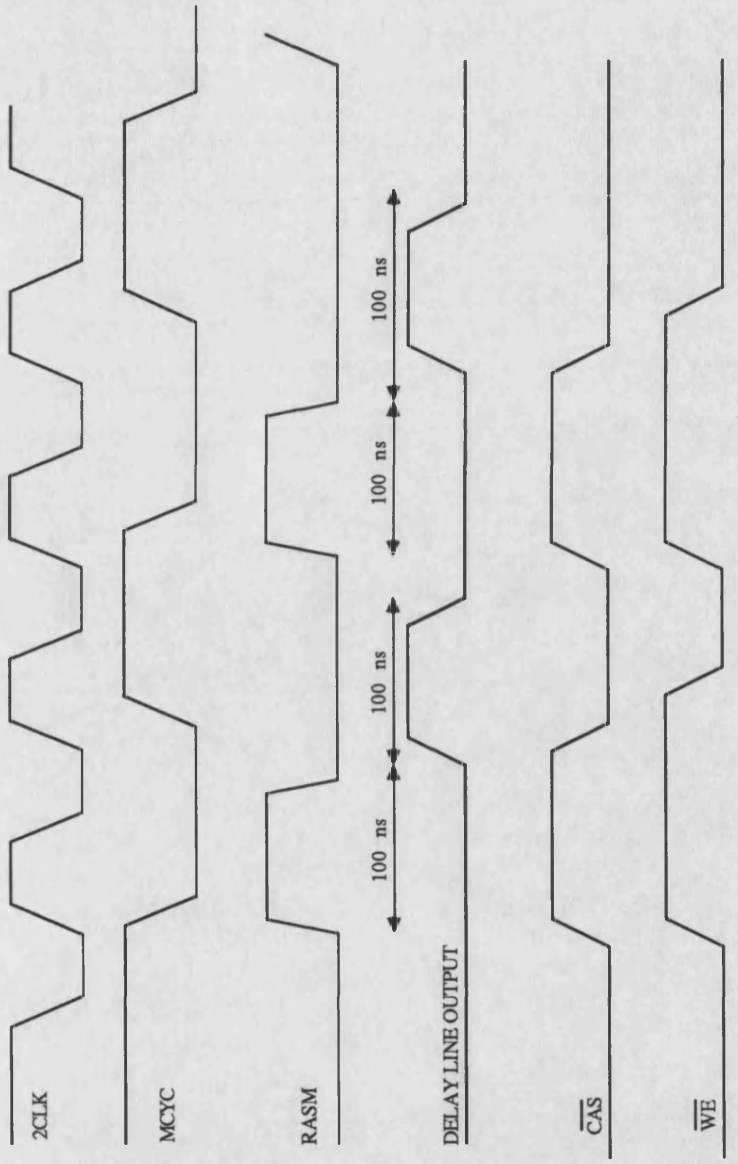


FIGURE 4.11 TIMING SIGNALS GENERATED BY THE CIRCUIT
SHOWN IN FIGURE 4.10

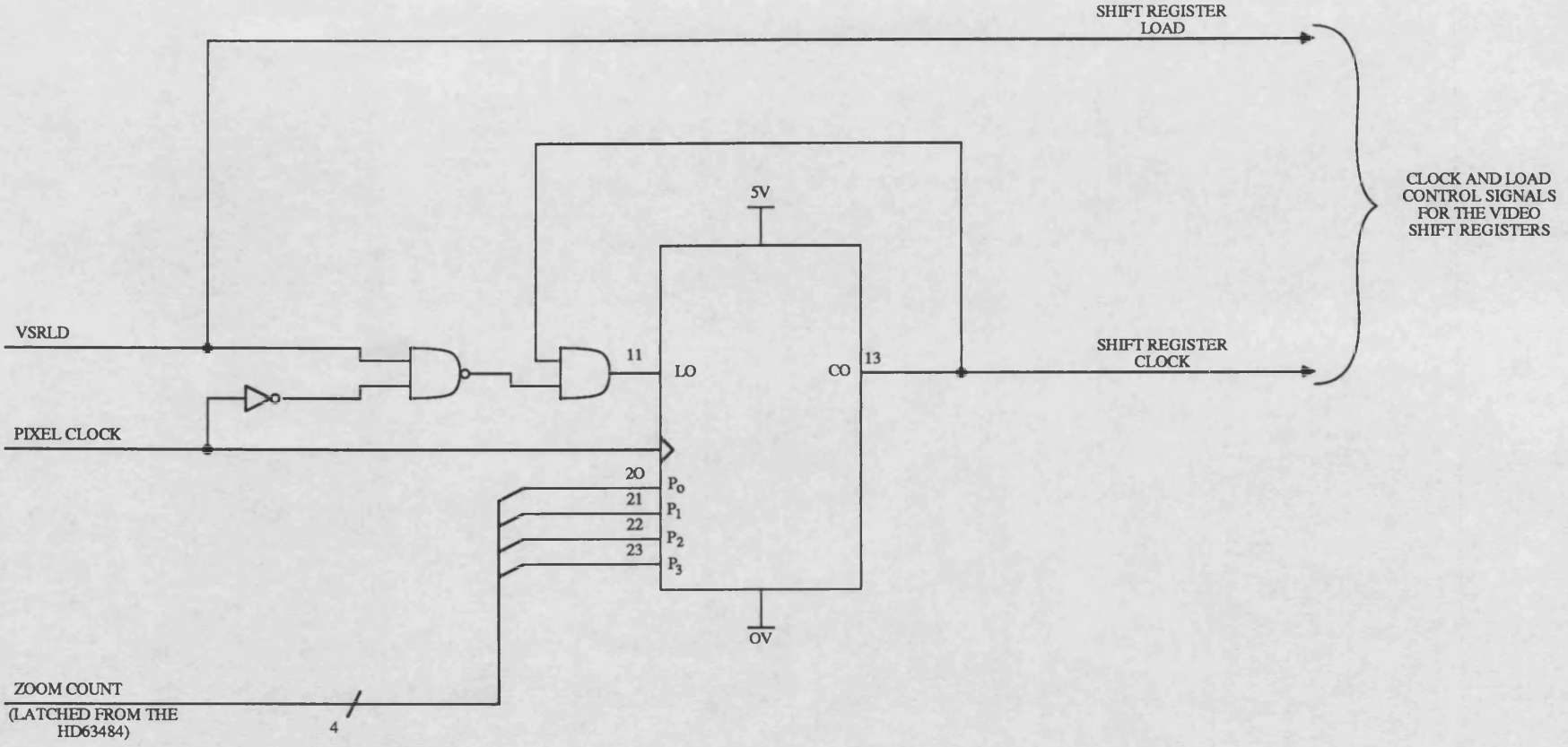


FIGURE 4.12 GENERATING THE VIDEO SHIFT REGISTER CONTROL SIGNALS TO PERFORM PICTURE ZOOMING

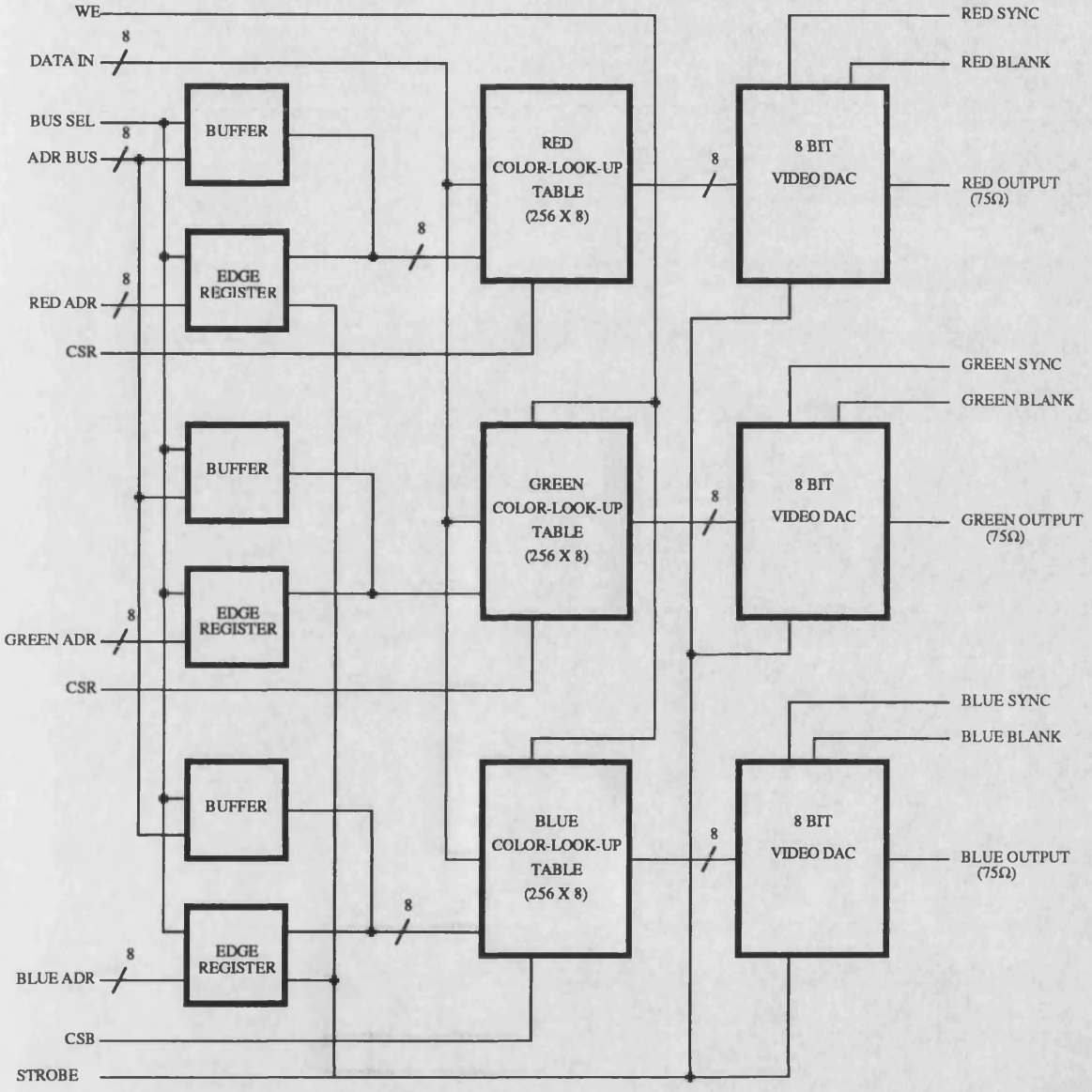


FIGURE 4.13 BLOCK DIAGRAM OF THE VIDEO DIGITAL TO ANALOGUE CONVERTOR

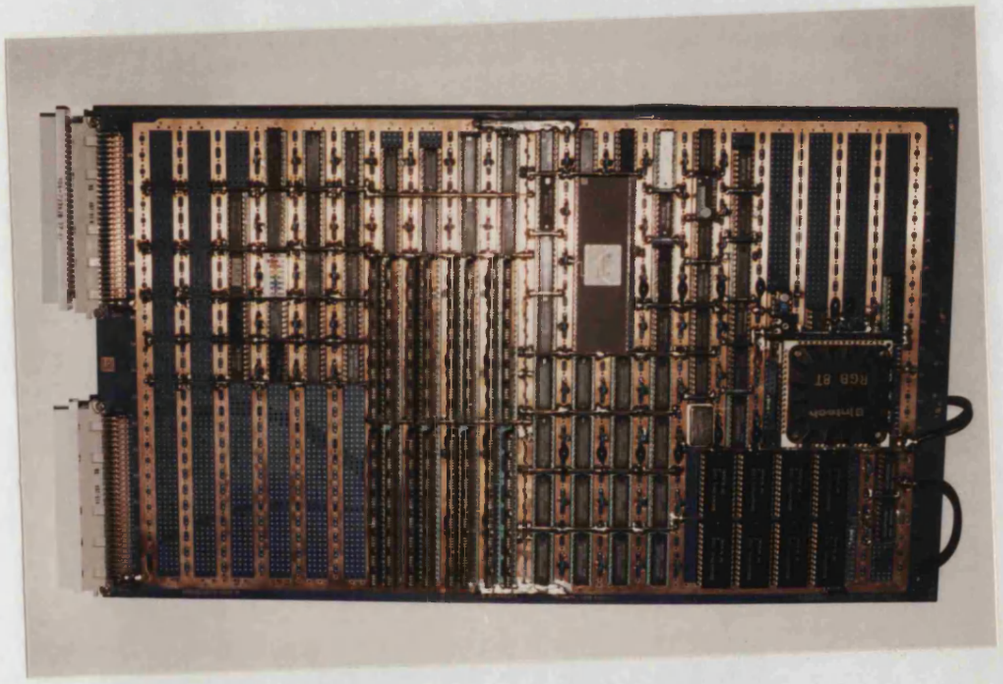


FIGURE 4.14 THE PROTOTYPE HD63484 COLOUR GRAPHICS SYSTEM

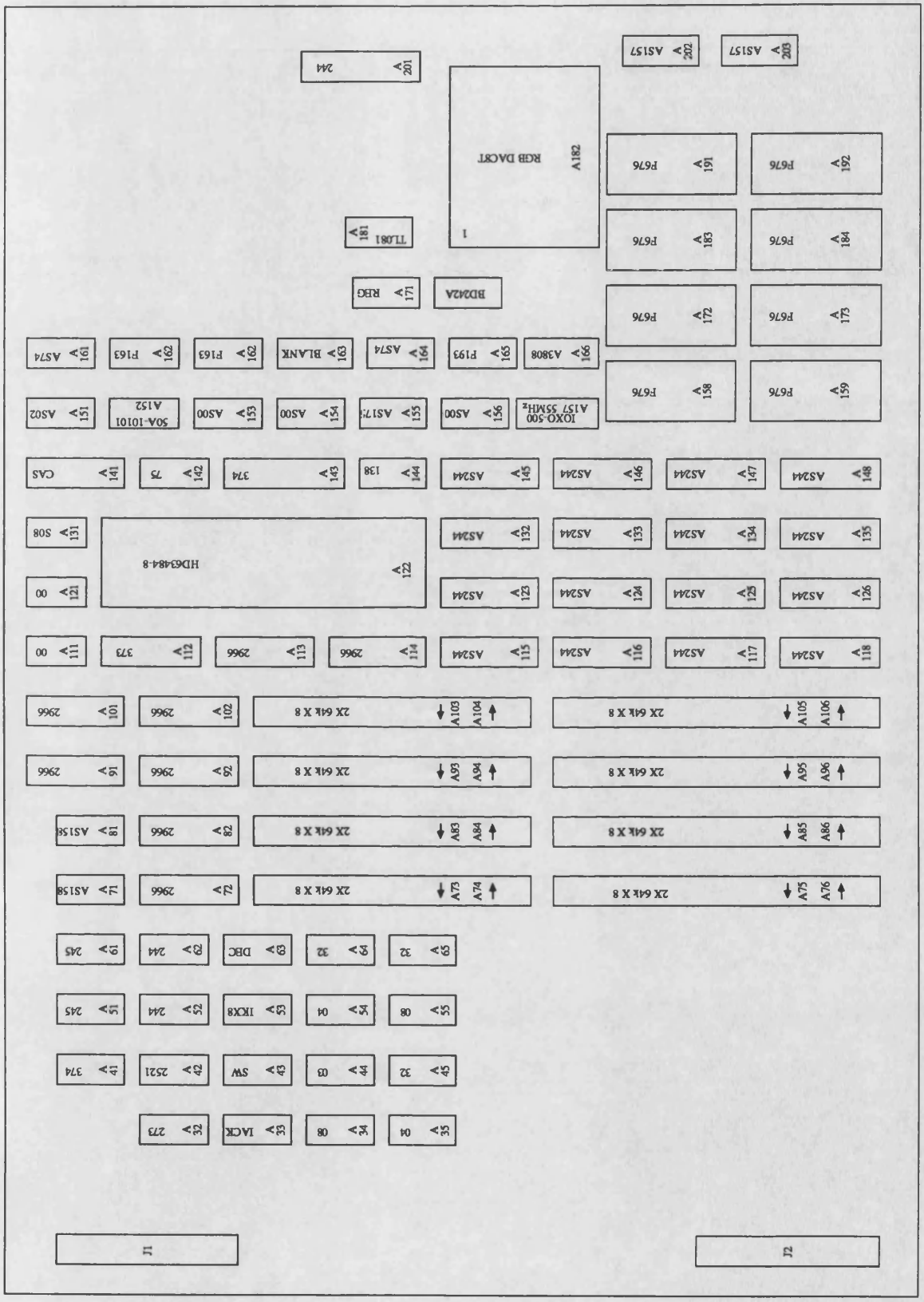
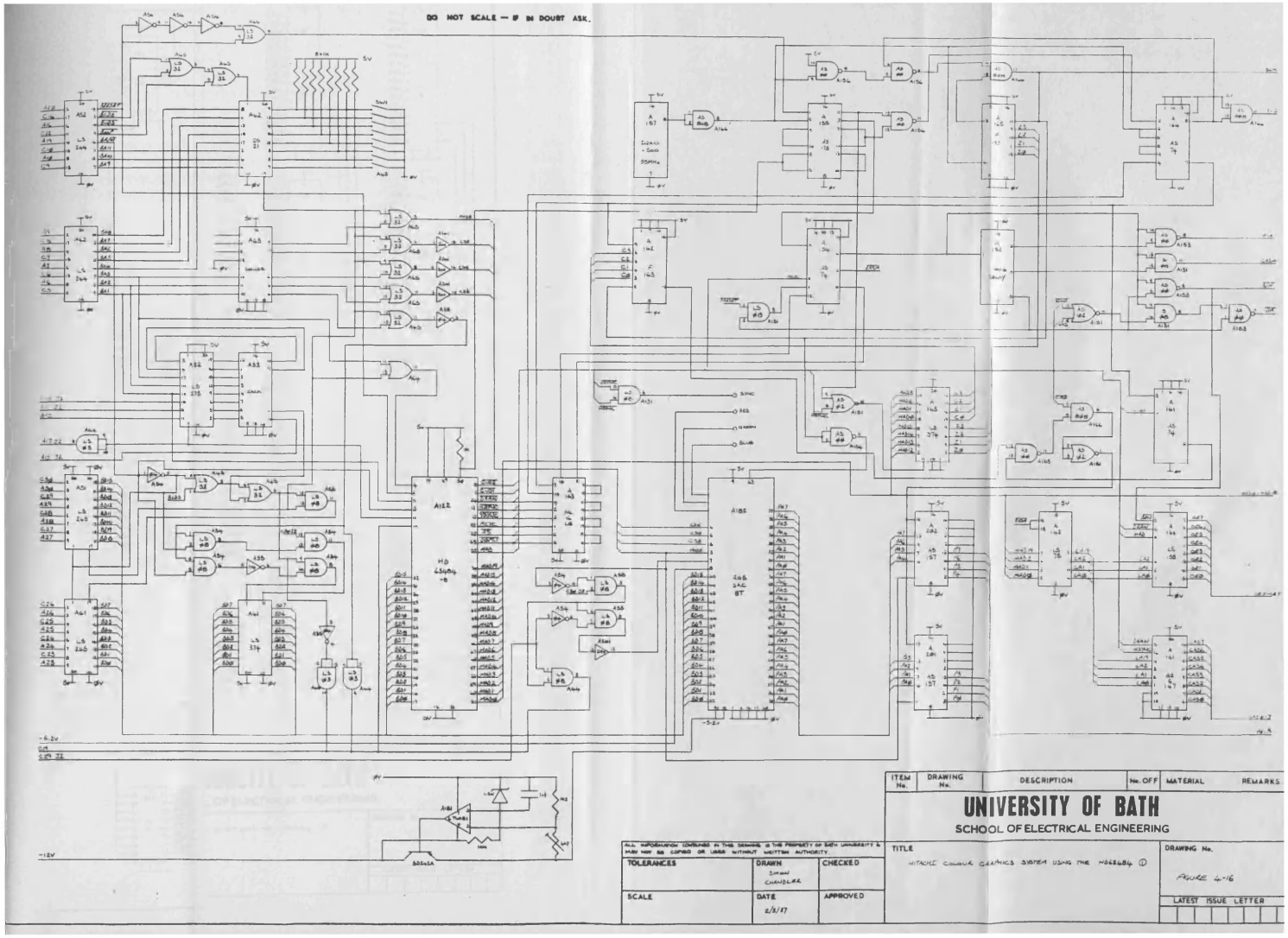
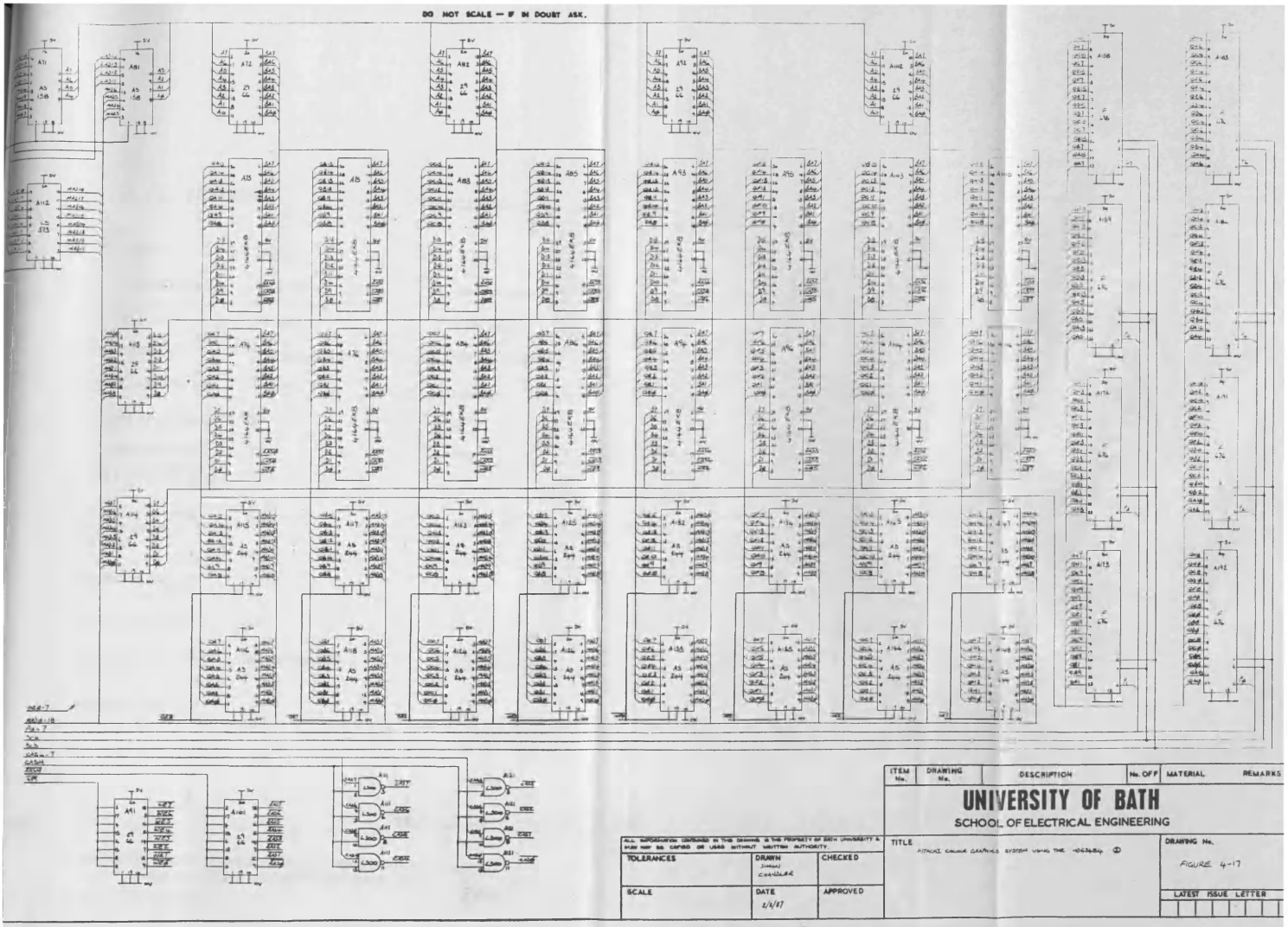


FIGURE 4.15 BOARD LAYOUT FOR THE HD63484 GRAPHICS SYSTEM


DO NOT SCALE — IF IN DOUBT ASK.
ITEM No.
DRAWING No.
DESCRIPTION
No. OFF
MATERIAL
REMARKS
UNIVERSITY OF BATH
SCHOOL OF ELECTRICAL ENGINEERING
ALL INFORMATION CONTAINED IN THIS DRAWING IS THE PROPERTY OF BATH UNIVERSITY & MAY NOT BE COPIED OR USED WITHOUT WRITTEN AUTHORITY.
TOLERANCES
DRAWN
SIMON CHANDLER
CHECKED
SCALE
DATE
2/3/87
APPROVED
TITLE
HITACHI COLOUR GRAPHICS SYSTEM USING THE HD6845S ©
DRAWING No.
FIGURE 4-16
LATEST ISSUE LETTER



ITEM No.	DRAWING No.	DESCRIPTION	No. OFF	MATERIAL	REMARKS

UNIVERSITY OF BATH

SCHOOL OF ELECTRICAL ENGINEERING

TOLERANCES	DRAWN	CHECKED
	JAMES CHANDLER	
SCALE	DATE	APPROVED
	2/4/87	

TITLE: HITACHI COLOUR GRAPHICS SYSTEM USING THE HD63484 ©

DRAWING No.: FIGURE 4-17

LATEST ISSUE LETTER:

	IGP SYSTEM	HD63484 SYSTEM
PIXEL CLOCK	28 MH_Z	55 MH_Z
USEFUL RESOLUTION	780 X 780	1024 X 1024
SYSTEM CLOCK	3.5 MH_Z	6.875 MH_Z
DRAWING SPEED	1.14 msec/PIXEL	580 nsec/PIXEL
DISPLAY MEMORY SIZE	1 M Byte	1 M Byte
NUMBER OF BITS/PIXEL	8	8
MAXIMUM NUMBER OF SIMULTANEOUSLY DISPLAYABLE COLOURS	256	256
MAXIMUM NUMBER OF DISPLAYABLE COLOURS	4096	16.77 X 10^6
HARDWARE ZOOM	1 - 16	1 - 16
HARDWARE CURSOR	YES	YES
HOST COMMUNICATION METHOD	SHARED MEMORY	FIFO BUFFER
INTERRUPT ON COMPLETION	YES	YES

FIGURE 4.18 HARDWARE PERFORMANCE COMPARISON OF THE NEC7220 BASED IGP SYSTEM AND THE HD63484 BASED SYSTEM

	IGP SYSTEM	HD63484 SYSTEM
LOGICAL X-Y ADDRESSING	√	√
VECTOR DRAWING	√	√
MARKER DRAWING	√	√
TEXT (PROGRAMMABLE CHARACTER SET)	√	√
AREA FILL	X	√
CICLE PLOTTING	X	√
ELLIPSE PLOTTING	X	√
LOGICAL OPERTION ON FRAME STORE DATA	√	√
PRIMITIVE CLIPPING	X	√
SOFTWARE PROGRAMMABLE CURSOR	√	√

FIGURE 4.19 FUNCTIONAL PERFORMANCE COMPARISON OF THE NEC7220 BASED IGP SYSTEM AND THE HD63484 BASED SYSTEM

CHAPTER 5

IMPLEMENTING GKS FOR TRIPOS

5.1 Graphics System Software.

A workstation for general purpose engineering and scientific applications must provide a device independent interface to its hardware facilities if it is to benefit from supporting commercially available CAD software. The first internationally accepted standard for computer graphics GKS, became the natural choice to ensure that this procedure was adhered to. An overview of GKS was presented in chapter 3, and this chapter describes the practical implementation details of GKS for the TRIPOS operating system written in BCPL.

5.2 A GKS Implementation for TRIPOS.

The implementation of GKS for TRIPOS, to be known as TGKS, is based on the specification in the British Standards Institution document, number BS 6390:1983[21], which describes the abstract function set and data structures associated with GKS. When work commenced on TGKS there were two programming languages available under the TRIPOS operating system, these being BCPL and FORTRAN 66. BCPL was chosen for the following reasons. Firstly it is the system programming language of TRIPOS and, as such, provides a very powerful link with the functionality of the operating system. Direct access to the kernel primitives that control communication between tasks and drivers allows very efficient data transfer to the physical graphics devices, which is particularly important in TGKS where data flow is high due to the low

level interface of the devices used. Secondly, the block structured nature of BCPL with its many looping constructs readily supports the implementation of graphics algorithms and improves the readability of the resulting code. Since BCPL is a typeless language, it allows great flexibility in modelling the abstract data types of GKS. A basic set of operators for integer and floating point manipulation are available for variables representing these types. Also available is a set of pointer operators to support memory indirection allowing an elegant and efficient implementation of the GKS internal data structures to be made. Finally, the use of the BCPL Global Vector for linkage between the application program and the GKS library of functions obviates the need for a separate linkage stage after compilation. The result is a faster production of application object modules which is particularly welcome during the debugging stages of program development. Also, smaller object modules are produced which can be loaded into memory faster and require less secondary storage space. The TRIPOS library mechanism[32] is used to load the TGKS library into memory prior to its use. During this process, the Global Vector is initialized with pointers to the TGKS functions, therefore making them publically available. The GKS library then resides in memory ready to be used by the application program, which is run as a co-routine to the users CLI task. Since co-routines share the same global vector, linkage to the GKS library by the application program is performed automatically.

5.2.1 A BCPL Language Binding for TGKS.

A language binding for GKS specifies how the abstract data types used in the GKS documentation are mapped onto the data types supported by the host language. It also specifies how the the abstract function names and argument lists of the document are mapped onto the host language function specification.

The features of the block structured language BCPL have affected how the binding has been implemented, resulting in differences to the F77 binding of ANSI[21].

There are six simple abstract data types defined by GKS. These are integer, real, string, point, name and enumerated type. Higher level data types are also specified, these resulting from a compound or combination of the lower level types. The mapping from simple GKS type to BCPL type was done in the following way. Integers and reals are straight forward, their use being controlled simply by their context. The GKS string type is modelled by the BCPL string handling mechanism, which packs the character values into BCPL words to give the most economical form of storage. The value of a BCPL string is a pointer to the vector containing the string. A GKS point is represented by a pair of BCPL pointers which are the addresses where the x and y values associated with the point are stored. This representation is easily extended to represent a compound of n points where the BCPL pointers are the addresses of x and y vectors of length n. The GKS name type is modelled by the BCPL string mechanism. Finally, the GKS enumerated type uses the BCPL manifest constant declaration to associate an integer constant with the GKS enumerated identifiers.

The BCPL compiler places no restriction on the length of function names and all the characters of a name are significant, so the abstract GKS function names were mapped directly onto BCPL function names. Words were delimited by full stops, and certain common GKS abstract words such as inquire were shortened to inq. Words such as OF were removed to keep the definitions concise. As an example, the TGKS binding transforms the GKS name "OPEN GKS" to "open.gks" and that of "INQUIRE SET OF OPEN WORKSTATIONS" to

"inq.set.open.workstations".

The consequence of chosing such a binding does result in some of the function names becoming unwieldy. For example, "SELECT NORMALIZATION TRANSFORM" becomes select.normalization.transform. This problem is, however, offset by the fact that subroutine names are close to the natural language and do not require the programmer to remember abbreviations. Annex C of BS 6390:1983 lays down several guide lines for implementing a language binding, and Rule L2, "A one-to-one mapping from language functions to abstract functions is preferred," was interpreted literally.

The GKS abstract functions are modelled by the BCPL function mechanism using the VALOF-RESULTIS construct to return a value to the caller. This value represents the error indicator of GKS and corresponds to that defined in the standard. If the function was successful, an error value of zero is returned to the caller.

Parameters are passed as arguments to the GKS functions, with the type representation described above. The unwary programmer may fall into traps, as the explicit lack of data type checking could result in unusual or obscure faults.

5.2.2 The Device Independent/Device Dependent Interface.

The application interface of GKS is independent of the physical devices it is driving, but, within any implementation of GKS there must be a point where processing becomes device dependent. The definition of this Device Independent/Device Dependent interface(DI/DD) was made very early on in the design of TGKS and was the result of many factors, including the physical devices that TGKS was to drive, and how easy it should be to add new devices

to the set that already existed. The placing of the DI/DD interface also had repercussions affecting how TGKS was implemented.

The function set of the DI/DD interface may range from only a very few simple graphical operations at the lowest level to almost all the GKS function set in very device specific implementations. The GKS standard introduces an abstract DI/DD interface called the workstation interface. The functionality of a GKS workstation was described in chapter 3 and represents that of an ideal physical device. If this interface is adopted, then the stage of defining another DI/DD interface is avoided. Although some manufacturers have produced graphical equipment that presents the host computer with a "GKS workstation interface"[50], they are expensive and it is very unlikely that TGKS would be required to drive such devices. When driving simple devices with a DI/DD interface at the GKS workstation level, considerable amounts of program code will reside in the device dependent section, making the addition of new devices a lengthy process.

For TGKS, a separate DI/DD interface was designed that reflected the functionality of the devices that it was reasonably expected to drive. When TGKS was implemented there were four physical devices available, these being the Thomson-Efcis EF9365 system[24], the IGP system described in chapter 4, a data terminal with TEKTRONIX emulation[51] and a pen plotter[52]. All these devices were capable of plotting points, drawing lines and drawing characters. They were also capable of simulating some of the geometric and non geometric attributes of GKS such a line type, character size and colour with limited accuracy. Since the method of producing attribute simulation in these cases is very device specific, it was decided that attribute information should be passed across the DI/DD interface so that the device drivers could exploit the features

of the hardware in the most efficient manner. Following these guide lines lead to a DI/DD function set of only six graphical operations and two utilities for device and internal TGKS table management. The six graphical operations are clear display, draw line, draw marker, plot character, update the colour look up table and inquire colour intensity which returns the red, green and blue magnitudes associated with the inquired colour number. A complete description of the DI/DD function set and its associated parameters is given in appendix F.

The low level DI/DD interface of TGKS is ideal for the devices it is required to drive. The function set is small and easily defined, meaning that new devices may be added quickly. A typical device dependent driver is less than 300 lines long. Because the drivers are small, there is no repetition of simulation code for unsupported functions such as area filling. The disadvantage with the DI/DD interface of TGKS is its inflexibility in efficiently driving "intelligent" graphics devices, since these will be treated as though they were dumb.

5.2.3 Storage Management

There are many dynamic data structures defined in the GKS standard. The workstation state list, for example, is allocated and initialized during an invocation of Open Workstation and then de-allocated during the matching Close Workstation. The internal memory requirement of GKS may conceptually grow without limit. The number of simultaniously open workstations, or quantity of segment storage required by an application program using GKS is also unknown before hand.

The data structures passed to GKS may also be of variable length and these will require modification internally by the implementation. Algorithms to perform the transformation of co-ordinates and clipping to windows, for

example, will require temporary storage for parameters.

The software simulation required by graphics systems to fill the void between their functionality and that of an ideal GKS workstation will also require unspecified amounts of temporary storage. The fill area algorithms for example[13] have a storage requirement that is dependent on the number of vectors comprising the figure, and the number of edges that a particular scan line cuts as it passes from one side of the display to the other.

When Open GKS is called, a parameter is passed from the application program to indicate the maximum amount of storage that GKS may use internally for that particular invokation. In this implementation the number passed to GKS was used directly to allocate a block of memory from the heap using the BCPL library routine GETVEC[32]. This block is returned to the operating system when GKS closes using the routine FREEVEC.

There are three routines to the storage management system, INITBLK, GETBLK and FREBLK. INITBLK is used to mark the first and last cells of the block obtained by GETVEC so that the algorithms GET and FREE may function correctly when first called. GETBLK uses a first fit algorithm to allocate blocks of variable size and, if required, will coalesce adjacent free blocks to obtain the specified amount of memory. If successful, it returns a pointer to the allocated vector and zero if it was not. The FREBLK routine simply marks the block as being free so that subsequent calls to GETBLK may re-use it.

5.2.4 The Internal Data Structures of TGKS.

The internal data structures defined in the GKS standard were described in chapter 3. They are all built from the six basic abstract types or a combination

of these, and are represented in TGKS using the rules defined for the language binding. The descriptive GKS data structures must become available after a call to Open GKS is made, but some of the state lists are of a transitory nature and their existence will depend upon the trail of GKS functions that have previously been performed.

To maximize throughput, all the GKS data structures are held in memory during the course of their existence. If, for example, a workstation is opened, its state list is set up in memory and remains resident until the workstation is closed when the list is freed and the storage space returned to the GKS heap. The data lists are all single dimension vectors which contain bit patterns representing either the simple GKS types directly, or a pointer to a higher level structure such as the bundle tables in the workstation description list. The offsets into each data list vector are associated with identifiers using the manifest constant declaration of BCPL. These identifiers were grouped together in a header file so that they could be made public to the TGKS subroutines and served to define the exact configuration that each vector should take. In this way, a complete definition of the data structures of TGKS could be defined before the coding of subroutines commenced.

All the descriptive information about TGKS and the workstation types it supports is resident within the loadable object code forming the TGKS library. Each TGKS descriptive list is initialized using direct assignment statements that are grouped into individual subroutines, one for each list. In this way, adding a new workstation requires the writing of a subroutine to initialize its description list. In practice, this is no problem as the subroutine has a well defined structure and may simply be copied from an existing workstation and the necessary modifications made. The new subroutine must then be added to the

TGKS library of subroutines, and a call to it inserted into the initialization subroutine of TGKS. A compilation and linking stage completes the necessary changes to add a new workstation description list to TGKS.

Other techniques were considered in an attempt to allow initialization data to be specified to TGKS in a more convenient manner. Such data could have been localized in a special format header file, but this would still require TGKS to be re-complied when making parameter changes. The optimum solution would have been to generate a description file that contained all the necessary information. This file could then be updated with parameter changes or information about new workstations without requiring a re-compilation of TGKS.

All the data structures of TGKS are linked together to form a tree structure. A path exists from the root of this tree to any piece of information that may be required by the internal routines of TGKS. Because BCPL supports operators to handle memory pointers and memory indirection in an efficient manner, data may be accessed very quickly by traversing the TGKS tree structure. A location in the BCPL Global Vector called the gks-pointer is reserved as the root of the tree, and points to the first data structure in this tree. Figure 5.1 shows the organisation of this structure. The first element is used to store the GKS operating state. The second element is a pointer to the GKS description table and the third points to the GKS state list. The fourth element points to the first workstation description table which is then linked onto subsequent description tables, so that this structure may be easily extended to accommodate new workstation types. The fifth element points to the first workstation state list which is again chained on to subsequent state lists, which may be linked and unlinked easily as workstations are opened and closed. The segment state list is

not used, but could point to a linked list of segments. The gks error list is pointed to by the seventh element of this initial structure, and the eighth element points to the GKS heap that was allocated using the buffer size parameter during the call to "Open GKS".

All the TGKS data structures are closely modelled on those defined in the GKS document, although for the workstation description table an additional set of pointers has been added. These are used to indicate the entry points to the device dependent functions for that particular workstation type. The management of these pointers is covered in more detail in the section on the device dependent routines of TGKS.

5.2.5 The Device Independent Layer of TGKS.

The device independent layer contains all the TGKS functions listed in the language binding[53], which may be split up into the following classes of, control, output, output attribute, transformation, segment, input, inquiry, utility and error handing functions. Each class of function starts with a phase of error checking, based on the specification of possible errors defined in the GKS standard. Inquiry functions will simply return any detected error as their value, but the other functions will enter an error logging phase which will report the error number together with its associated error message. Many functions simply update the internal TGKS data structures, or return the user information contained within them. The Set Colour Representation function also updates the colour look up table of raster scan devices.

The positioning of the DI/DD interface has forced all of the GKS primitive viewing pipeline processing up into the DI part of TGKS. A two stage transformation is performed for the coordinates between World

Coordinates(WC) and Normalized Device Coordinates(NDC) and then from NDC to Device Coordinates(DC). A positional clipping algorithm is used for the Polymarker and Text primitives. A vector clipping algorithm[8] is used for the Polyline primitive, and a polygon clipping algorithm[9] is used for Fill Area.

The functionality of the device dependent interface dictates what simulations will be required in the DI layer to map GKS primitives to device dependent primitives. The Polyline and Polymarker functions were easily produced, but additional processing is required for Fill Area, Text and Cell Array.

Area filling was performed by a scan conversion algorithm[13] which generates horizontal lines to completely shade the inside of a bounded polygon. The algorithm complies with the GKS specification for area filling, which states how self intersecting polygons should be filled. The coordinate input to this algorithm comes from the output of the viewing pipeline and works in the device coordinate units.

The cell array simulation provides the minimum requirement of GKS which is to draw the boundaries of the individual cells. The line drawing routine is again used to produce the required output.

The text primitive of GKS has 4 geometric, and 4 non geometric attributes affecting its appearance. Only the size, colour and text direction attributes are passed across the DI/DD interface, and additional processing is necessary to ensure that the text output produced by the limited quality character generators of the physical graphics devices match that requested as closely as possible. No stroke precision text is available in TGKS.

5.2.6 The Device Dependent Layer of TGKS.

Two methods of passing control across the DI/DD interface were considered. The first, known as the subroutine interface, consists of an entry point for every device dependent function required, with the data being passed as parameters to these subroutines. The second method, called the data interface, consists of a single subroutine, with the device dependent function being encoded and passed to this routine together with the appropriate data that has been packaged into some standard form. This subroutine then decodes the function and unpacks the data to perform the required function. A subroutine interface was implemented for TGKS because the data packing and unpacking stages could be avoided and the very small number of functions required by the device dependent routines would not become cumbersome.

A subroutine interface could be difficult to implement if multiple workstations were to be driven but the function pointer facility of BCPL enabled a simple and efficient method of calling the device dependent routines. The workstation description table has been extended in TGKS to hold the pointers to the device dependent routines for that workstation. Since these routines are called by reference to the workstation description table, their actual names are unimportant which means that no modification to the device independent code is necessary when adding new graphics devices to the system. The device dependent routines for each workstation are held in individual files, the names of which permit them to be identified with the corresponding workstation name. When a workstation is opened, its device dependent routines are loaded from the file and the routine pointers in the workstation description list are set up. A use count is kept on the device dependent routines to prevent them being multiply loaded by simultaneously open workstations of the same type. When the last workstation of a particular type is finally closed, the use count for its device dependent routines falls to zero and they are unloaded

from memory. A single global variable is used to point to the device dependent initialization routine. When the device dependent code is first loaded, a global initialisation is performed to set up this global pointer so that the initialisation routine may be called. It is this routine that sets up the pointers to the other device dependent routines in the workstation description list. Since the initialise routine for a particular workstation is called once only, its global variable may be shared between all the workstation types.

The six graphical routines supported by the device handler may be responsible for a certain amount of attribute simulation, but only in as far as exploiting the features of a physical device to obtain an output that is as close as possible to that requested. A good example of this is the setting of character size using the zoom drawing feature of the THOMSON-EFCIS system. Devices with their own local intelligence, such as the IGP system and the pen plotter, require no simulation and may be passed the attribute values directly.

The graphics devices supported on the TRIPOS system fall into two categories. The first of these contain devices that communicate with the host computer using an RS232 type serial connection, such as the pen plotter. These devices necessarily support a data transfer protocol that is character based, being easily implemented using the standard character input and output routines of the host programming language. The lower level hardware features of the RS232 interconnection are shielded from the TGKS device dependent routines by the facilities of the programming language, so that writing device dependent routines for character based protocols is straight forward. For TGKS, the BCPL library routines for character input and output were used to communicate with this class of device.

The second class of devices are those that communicate directly with the

microprocessor using the backplane bus. These devices are memory mapped, and could be directly accessed using BCPL assignment statements. It is, however, more efficient to operate them as TRIPOS devices, using the method described in chapter 2. In these instances, the device dependent part of TGKS is made up of two sections, one written in BCPL and the other in assembly code. The BCPL section contains all the entry points for the DI/DD interface and the assembly code section contains all the routines required by a TRIPOS device. The TRIPOS packet passing mechanism is used to communicate between the TGKS task and the TRIPOS device. Hence, packets may be queued to the device as fast as they are generated, whilst the device processes the packets at its own speed under coordination from its own interrupt routine. In this manner, the multi-tasking ability of TRIPOS can be used to improve throughput, as there is no waiting time associated with the device communication process. The assembly code section of a TRIPOS device driver may be loaded and unloaded as it is required, and this task is performed by the device dependent initialise and uninitialize routines respectively. The drivers are not an integral part of the TRIPOS kernel and therefore do not require a customized kernel to be produced for each new device. The time to produce a new driver is considerably reduced by this feature, when compared to other operating systems such as UNIX which include their device drivers in their kernel.

5.2.7 Applications for TGKS.

The TGKS library currently fulfills the functionality requirement of a level Oa implementation of GKS, that is, basic output and no input support. At this level there are 79 user callable GKS subroutines. TGKS has been used by undergraduates for project work on orthodontic viewing techniques[54], and connectivity studies[55], and has also been used by postgraduates exploring

three dimensional graphics[56] and for a graph plotting package[30]. These applications have shown that TGKS has the necessary power and speed to be used in a wide range of passive graphics applications. In its present form, TGKS consists of approximately 3000 lines of BCPL source code and is capable of driving four raster scan graphics systems and one pen plotter. TGKS was to be the basis of the graphics package for the computer graphics node described in chapter 4 until a switch to UNIX was made in an attempt to construct a single board workstation system running the Rutherford-Appleton implementation of GKS, called RAL GKS. A description of this system and its development are given in chapter 6.

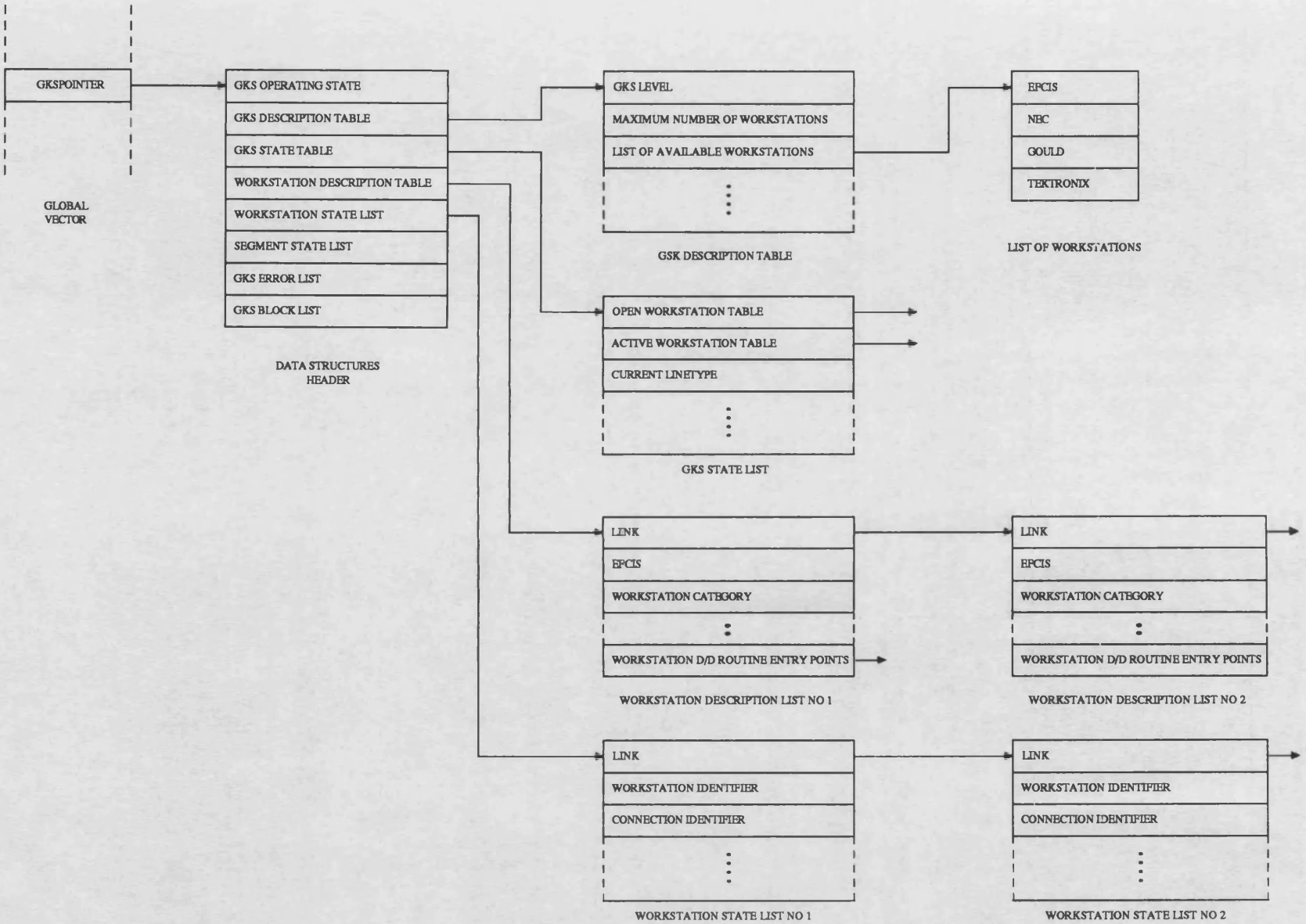


FIGURE 5.1 THE TGKS DATA STRUCTURES

CHAPTER 6

A UNIX WORKSTATION SUPPORTING GKS

6.1 A Single User UNIX Workstation.

When TGKS had reached the GKS level "0a" functionality, an appraisal of the possible avenues leading to the development of a general purpose, single user workstation were made. Extensions to TGKS could have been made to support segmented displays and input devices, and hence, a powerful environment for running CAD applications could have been constructed. There was, however, another possible route created by the availability of the UNIX operating system running on the Single Board Computer(SBC). Although the TRIPOS system offered the advantages of being small, efficient and having extended semantics to support multi-processor systems, its lack of support particularly for FORTRAN 77(F77) and its relative obscurity amongst industrial users resulted in the UNIX operating system being adopted for this workstation. UNIX is widely accepted amongst scientific users and is increasingly being applied to business markets.

When the hardware development of the Hitachi HD63484 system was finished, the only component requiring further development was an input device interface to allow the SBC workstation to communicate with devices such as mice and joysticks. With this addition, a complete systems solution was available to produce a single user workstation running the UNIX v5.2 operating system. The SBC described in chapter 2 was complimented by the graphics system, together with an additional two megabytes of RAM and the Multi-Link local area networking system. An 85 megabyte hard disc system

was used as the secondary storage media.

To complete the workstation configuration, a BBC computer running a data terminal emulation program was used for the operators console. The facilities of the BBC computer were extended to enhance the interactive capabilities of the system by supporting graphical mouse and joystick input devices for GKS.

With the hardware in place, the next phase of design involved the development of the systems software to support graphics and CAD applications. The widespread use of FORTRAN for CAD applications dictated that a GKS implementation supporting the F77 language binding[21] should be used. The effort required to produce a complete F77 implementation ruled out this approach. Instead, a GKS system was ported onto the SBC workstation, and software written to allow its use with the HD63484 graphics system and a GOULD pen plotter.

6.2 GKS Input Model Simulation Using a BBC Computer.

Earlier work on interactive techniques using the TGKS system[56] had involved the development of an interface card providing memory mapped Analogue to Digital converters and parallel ports for the MC68000 system. The A/D convertors were used with a joystick control to give X and Y coordinate inputs, and the parallel ports were used for switch type input. Although this system was successful, it did place an additional processing burden on the main processor. Better performance and throughput could be achieved by employing an I/O processor dedicated to this task. If backplane mounted input devices had been used, they would have required their own UNIX drivers to be written together with simulation software to map the GKS logical device model onto the physical devices.

A serial data link complying to the RS432[57] standard is supported by the BBC Computer for communication with the host computer. In general, this link is operated at 9600 baud.

In keeping with the aims of distributing as much of the graphics overhead from the main processor, it was decided to use the BBC computer as a front end processor to handle several input devices. The BBC computer was to run an extended version of the standard terminal emulator and a communications protocol defined so that the SBC workstation could control the input devices connected to the BBC. Strings of commands would request values from these devices or initialize the BBC to some state prior to an input request. The BBC computer is very flexible, and has a powerful operating system that supports many calls to control the physical devices that are available in its memory map.

Any additional processing, such as retrieving data from the workstation state lists upon device initialization, or floating point manipulation for transformations would be handled within the workstation. If required, echoing on the workstation display surface would also have to be done by the host microprocessor. The BBC interface would be very uniform however, and the peculiarities of the physical input devices connected to the BBC would be shielded by the simulation software it was running.

For maximum flexibility, it was proposed that each device class of GKS should be supported by several physical devices. Before work commenced on the BBC Input Emulator, the keyboard cursor keys had been used to provide graphical input. The UNIX workstation would poll the BBC Computer and update X and Y registers depending upon which cursor keys were pressed. Other routines were then used to provide a cursor echo feedback for the

operator. If the operator pressed the carriage return key, then a successful input was signalled to GKS. The use of the cursor keys of the BBC Computer were maintained in the input emulator, but the keyboard device was treated as an input device in its own right, and had an identical control protocol to the other input devices. Hence, the keyboard could be used to give input values for all of the supported GKS classes in the same way as more conventional input devices, and could also use the same software interface. Two other physical input devices were attached to the BBC computer so that their performance could be assessed in this application. The first of these was a joystick control with three axes of movement. The X and Y axes were used for conventional positional use while the Z axis was used for VALUATOR type input. After succesful operation with the joystick had been achieved, a mouse device was added. Both of these devices also had push button switches which were used to either terminate the request for input and return a value to GKS, or to signal a break operation with no returned value. The BBC computer also supports light pen input, but this was thought to be unnecessary with the devices that were already available. There are four analogue input channels on the BBC computer which provide A/D conversion to a maximum accuracy of 12 bits each. Three of these channels were used to interface to the joystick control. The mouse was connected to the BBC user port which provides a very flexible interface through the use of a Versatile Interface Adaptor(VIA) device[58]. In this application, the VIA was programmed to provide a parallel input with automatic interrupt generation to signal changes on the port inputs.

6.2.1 Developing ROM based Software for the BBC Computer.

The salient features of the BBC computer include a 6502 microprocessor[59] with multimode graphics supported by a HD6845 CRT controller. A Versatile

Interface Adaptor, four channel D/A converter, floppy disc interface, Light Pen interface and software switchable selection of Read Only Memory(ROM) devices for language and utility functions. The switchable ROMs are known as language ROMs, and the terminal and input emulation software is targeted to run as a language ROM.

The memory map is logically split in its centre with ROM and memory mapped peripherals in the top 32K bytes and RAM in the lower 32K bytes. The operating system resides in the upper 16Kbytes of the top split, and provides utilities to access the peripherals. It also allows selection of up to 16 different 16K language ROMs by mapping them into lower half of the upper split.

The BASIC interpreter ROM also contains an assembler, and this was used to generate the 6502 program code. A network system allowed file serving facilities from a host UNIX computer to be used for storage purposes. A commercially available expansion board for the BBC computer that provided a sideways RAM option was plugged into one of the BBC Computer language ROM slots. The assembled programs were loaded into the sideways RAM from where they could be tested as if they were resident ROM code. When a complete working system had been written in this way, the code was blown into an EPROM which was installed in place of the sideways RAM. The entry point to this modified terminal emulator is called from the BBC operating system in the conventional way for language ROMs[60]. It has retained its original name of "TERMINAL".

6.2.2 The Input Function Specification.

The logical input device model of GKS was described in chapter 3, together with the different classes and operating modes that a logical device may operate

in. The major factor preventing complete decentralized simulation of the GKS input model is the echoing of measure values when an input device's measure process is active. Although the standard does allow an echo window to be specified on the workstation display surface, the operation of certain device classes may be such that this echoing is unnecessary. For input that is truly graphical however, an echo on the display surface is mandatory and, in these instances, complete decentralized simulation would require that the input processor address the cursor echoing facilities of the display device directly. Communicating with the BBC computer using an RS432 connection can therefore never fulfil this aim. Instead, the facilities of the BBC computer could be associated with an input process running on the workstation. This process could act as a filter for incoming values from the BBC computer, trapping those for echoing, and passing others directly to the main GKS process. For REQUEST mode input, maximum use of the BBC computer would be made in the simulation of device classes for GKS and, where possible, the echoing of values on the screen would be removed by the representation on the simulated device. An input handler would be written in C to complement the functions provided by the BBC computer to approximate the action of a GKS logical device operating in request mode. The parallelism available in REQUEST mode would be harnessed by removing the need to deal with the specific hardware details of input devices from the workstation, and also through the simulation of GKS device classes by the BBC Computer.

Several areas were identified where the BBC computer could best be exploited. For Locator and Stroke simulations, the echoing of values to the workstation must be continuous during the period that the REQUEST operation is active. The viewport specification, and viewport priority will specify the range of valid input values that may be returned. Instead of testing the validity of the

values, a mapping within the BBC Computer may be used to automatically limit the range of output values to those of the viewport automatically.

The CHOICE and VALUATOR device classes allowed a considerable amount of simulation to be achieved by the BBC Computer. These classes provide non-graphical input and hence allow the BBC computer to give echoing, or device simulation of a type that does not require further echoing on the workstation display surface. Once a device of this type has been initialized, and assuming that no echoing on the display surface is required, the workstation needs perform no further input processing until the operator indicates a succesful input request or a break action.

For the string type input, no simulation was required by the BBC computer since character input was supplied by the original terminal emulation which also gave an echo of the keys pressed by displaying them on the terminal screen.

The keyboard, joystick or mouse input devices could all be used to return values to an application program in each of the GKS device classes. The initialization sequence protocols are the same in all cases, with the exception of the device identifier character. Since the BBC Computer offers device independence in this way, the driver software in the workstation is straightforward and uniform across different physical input devices.

A character based protocol has been designed to transmit data between the BBC computer and the UNIX workstation. Since coordinates are ten bit values, two characters are required to transmit a single value, the first character containing the upper 5 bits and the second character containing the lower five bits. The upper 2 bits of each character have at least one bit set to ensure that

control characters cannot be confused with coordinates. The upper two bits are also used to indicate which character in the coordinate sequence is being sent.

When the input simulator was first tested, it was programmed to send values to the workstation only if a change was noted in the input device. This reduced the input processing required by the workstation and helped prevent the potential loss of characters due to buffer overrun. This method was satisfactory for the joystick control, but when the much faster mouse device was used, its coordinate production rate was fast enough to cause character loss, and a subsequent loss of communication synchronization.

To overcome these problems, a handshaking protocol was introduced to allow the workstation to inquire the current state of any active input device. This protocol was extended to work with all the simulated device classes and enables GKS sample operation to be readily supported. When operating in this mode, the BBC computer was conceptually running the input emulation and terminal emulation programs in parallel.

The protocol is such that the workstation must request a value from the BBC Computer by sending a unique request control code. For mouse input the response was adequate to maintain a close resemblance between the motion of the mouse and the cursor position but for the joystick, a time lag was introduced which caused its echo to lag behind the actual position of the input device. The conversion time of the A/D converters was identified as the cause of the lag between the BBC Computer receiving the request control code and the time when two new coordinate values became available for sending to the host. Two flag registers were introduced to overcome this problem. One flag signifies that it is clear to send data to the workstation, i.e. that the workstation has sent a "request to send" code. The other flag register signifies that a D/A

conversion is complete and that the BBC Computer is available to send data. If a "request to send" is received from the workstation, and the available to send flag is set then a value will be sent immediately, otherwise the clear to send flag will be set and, when conversion is complete, the value will be sent. Whenever a value is sent to the workstation, both flags are reset, and further A/D conversions may commence.

A complete function specification and communication protocol for the input emulator for a BBC computer is given in appendix G.

6.2.3 Simulation for Locator, Stroke and Pick Input Classes.

The joystick control could return values with a 12 bit accuracy, but in practice this was not necessary since the workstation was only capable of displaying the cursor to a 10 bit accuracy. Initial operation with the joystick showed a more serious limit on the accuracy that could be achieved. This limit was imposed by the noise generated in the joystick potentiometers and the A/D conversion process. Noise caused the cursor position to jitter, and increased the number of samples that were being transmitted back to the UNIX workstation. To overcome this problem, a new input value was only logged if it varied from the previous one by more than 4. The reduction in accuracy obtained by this crude filter caused no problems in practice.

A second method of removing jitter involved the implementation of a velocity type joystick. The same physical device was used, but instead of its position representing the screen position, its displacement from the centre stalled position represented the velocity of the cursor in that direction. Although this worked well in principle, the lack of a sprung return to the stalled position in the joystick meant that it was hard to stop the cursor

moving, and consequently hard to pick items using this type of device.

When using the mouse for positional input, the jitter problem was not present. The mouse contained two slotted discs which were rotated in sympathy with the motion of the mouse. The slotted discs generated pulses representing approximately 0.25mm of motion in either the X or Y direction. The incremental precision of the mouse device was chosen to allow a reasonably sized single sweep with the mouse to traverse the whole display. A sweep of approximately 20cm will therefore cause the position registers to pass through the complete range of 1024 values.

6.2.4 Simulation for Choice Input.

The BBC computer was used as a programmable choice device, allowing a maximum of 16 menus to be stored simultaneously, each to a maximum of 32 options. The BBC VDU was used to display the choice strings which may be up to 8 characters in length. Each option was displayed centrally within a bounding box, with immediate echoing of the current choice by underling. The BBC VDU mode used to display the choice menu did not permit graphics, so a character set had to be defined from which the bounding boxes and enhanced underlining could be generated. The display was configured so that even when the full set of menu boxes was used, there was at least 10 character lines still available for conventional text output at the bottom of the screen. This available space is scrolled automatically if output is generated by the workstation during the course of a choice interaction, so that the choice display is not corrupted. The logical prompt is the appearance of the choice menu, the echo is the movement of the highlight underline between boxes, and the acknowledgement is the disappearance of the choice menu after the operator has pressed either the input or break action buttons on the physical input device.

6.2.5 Simulation for Valuator Input.

Valuator input is based on many of the subroutine functions developed for choice input. The operating principles are the same, but the echoing method involves the display of a linear scale with programmable markers that consist of text strings of up to 8 characters. A line marker is used to echo the value, which is returned to the workstation as a ten bit value which must then be scaled to reflect the maximum valuator range.

6.3 An Overview of RAL GKS.

An implementation of the Graphical Kernel System has been produced by Rutherford Appleton Laboratories(RAL). The work was done as part of a collaborative effort with ICL. RAL GKS fulfils the functionality requirement defined in the GKS standard as level "1b" and represents approximately ten man years of programming effort. The team involved with this implementation also participated in the graphics standards working parties whose aim was to produce the specification for GKS.

Portability was of prime importance to Rutherford. They chose FORTRAN77, as this is the most widely accepted scientific programming language, particularly amongst industrial users. The popularity of FORTRAN has made it almost mandatory for computer manufacturers to supply a compiler for this language on their machines. Unfortunately, FORTRAN is not a good systems programming language since it has evolved to meet the needs of the scientific user. It does not permit the efficient and elegant constructs that languages such as BCPL or C support.

The language binding for RAL GKS is taken from the DIN FORTRAN binding[21]. Routine names begin with a "g" and are followed by the

concatenated and abbreviated forms of the abstract GKS function names. The DI/DD interface is placed at the GKS workstation interface, hence, considerable simulation software is required for workstations that employ dumb devices. A data interface is used to pass control across the DI/DD interface. Considerable support in the form of FORTRAN simulation subroutines for functions such as stroke text font generation and area filling were supplied. A GKS workstation is therefore implemented as a single large subroutine typically consisting of over 1000 lines of source code.

Segmentation is supported and utilities are provided to place segments in a Central Segment Store(CSS) for devices that are not capable of storing their own segments. Future extensions to RAL GKS could use this central segment store to implement the Workstation Independent Segment Store(WISS) of GKS.

Metafile input and output workstations are also implemented, using the format specified in annex E of BS6390:1983[21].

The choice of F77 for RAL GKS allowed other programming languages to access the GKS subroutine library. On the SBC workstation, object modules produced by the C compiler and the Fortran compiler could be made compatible. Hence, application programs written in C could use RAL GKS, and certain functions of the workstation driver for the Hitachi HD63484 system could be written in C.

Adding new workstations to RAL GKS is a straightforward task, requiring two components. The first is a text file containing descriptive information about the workstation and the second is the workstation driver subroutine, whose entry points and functions satisfy those specified for the RAL GKS DI/DD interface[61]. The workstation driver subroutine is given a special name which

must also be included in its description file. The descriptive text file is concatenated with the other workstation description files to produce an output file that can be used by a pre-processor to convert it into a FORTRAN direct access file. This file is then used by RAL GKS to update its "in core" information about the workstations. An F77 "INCLUDE" file is also produced by the pre-processor. This file contains statements which conditionally route control to the various workstations depending upon their state. The names of the workstation drivers are taken from the workstation description files. The six subroutines that call the workstation drivers must be re-compiled using this "INCLUDE" file, which is placed at the end of their text segments. When these six routines and the new workstation driver have been updated in the GKS library, the task of adding a new workstation is complete. Unfortunately, because the GKS library must be updated with the new driver each time that it is altered, testing a new driver is a very slow process, due mainly, to the time taken to link the test application program to the RAL GKS library which necessarily contains a great many subroutine entries.

Direct access files are used for the workstation description tables, error message tables and text font description tables. Utilities were supplied by Rutherford to translate readable text files into suitable direct access files.

6.4 Porting RAL GKS onto the SBC workstation.

Porting the RAL GKS library of subroutines onto the SBC workstation should have been a very straight forward task because all the F77 constructs that are associated with specialized dialects of the language have purposefully been avoided. A small set of operating system specific subroutines were well documented so that they could easily be implemented. One of these, a subroutine to obtain the date from UNIX, was implemented in C, and made

compatible with the F77 interface.

All the RAL GKS subroutines that used "INCLUDE" files needed to have their file name specification modified to agree with the UNIX file naming convention. A shell script[37] was written to automate this process, and was also used to convert upper case characters to lower case, which is the format most suitable for the F77 compiler.

Porting Rutherford GKS onto the SBC UNIX system showed up several bugs in the Fortran Compiler. Such bugs were not seen when tested on other UNIX machines, so must be put down to our own UNIX/hardware combination. These bugs considerably lengthened the porting exercise as changes to the code had to be made to overcome them. Had these bugs not been present, no alteration of the code would have been necessary at all.

When the bulk of code from Rutherford had been tested, a workstation driver subroutine was written to allow RAL GKS to drive the HD63484 graphics system and BBC input emulation system as a combined OUTIN type workstation. A driver was also written for the GOULD pen plotter as an OUTPUT type device which could be used by an application program to obtain graphical hard copies.

6.5 A GKS Workstation Driver for the HD63484 Raster Graphics System.

A workstation driver for RAL GKS must provide all the functions expected of an abstract GKS workstation. The Hitachi HD63484 system is capable of fast graphical output and supports a high level programming interface that is suitable for generating the GKS primitives. It is however, not capable of performing segment storage, coordinate transformation and state list management. As a consequence, the driver code for the HD63484 system is

large, but it has been tailored to exploit the power of the HD63484 to maximize throughput.

The elegant structure of UNIX permits all I/O devices to be addressed in a uniform fashion. All devices including the terminal I/O channels and the Hitachi device are accessed as though they were special files. All special files have the same well defined interface that contains only a few simple operations. These functions include operations to open, read, write and close a device. The routines which handle these functions are part of the kernel, the development of which is a four phase process. Firstly, the driver software is developed on a source UNIX machine that has all the constituent parts of the kernel available. The kernel must then be built, moved to the development machine and finally debugged. The environment for software development and debugging is strictly limited once operating in the kernel and this can frustrate development effort. One bonus has been that the driver is written in 'C' which has improved development time and allowed higher level functions to be included in the driver. The latter has provided a more uniform driver interface and reduced the amount of data transferred during output requests to the HD63484 system.

It was proposed that the maximum use of the special features provided by the Hitachi device should be made when constructing the workstation driver. This driver has a three layer hierarchical structure, with each layer performing more device specific tasks as control passes downwards. At the top, there is the FORTRAN workstation interface to the RAL GKS front end. Using the RAL naming convention for workstation drivers makes this routine's identifier "gk0hwd". At the next level, there is the Hitachi specific handler which is written in C. A subroutine interface is used to pass control to this layer. At the

lowest level, there is the HD63484 driver which is part of the UNIX kernel and is, therefore, accessed using the standard input/output kernel entry points. The interface between the FORTRAN top level and the C handler subroutines is easily achieved provided that their names have the correct form and that their parameters are passed as specified in the UNIX documentation[36].

The RAL GKS system was supplied with a workstation driver for a TEKTRONIX type device. This driver was used to test the system during the porting stage, and was used to control a Data-Type terminal running a TEKTRONIX emulation. Since the TEKTRONIX interface in this case is very simple, consisting only of line drawing functions, the whole of the F77 simulation routines could also be tested. When satisfactory operation had been achieved in this way, a simple UNIX driver for the HD63484 was written that would provide initially only line drawing capability. With only minor modifications to the TEKTRONIX workstation driver, a Hitachi workstation driver was produced, and arranged in the three layer structure described above.

When this system had been tested, an optimal configuration was constructed to make the best use of the hardware features of the HD63484. The final functionality split between the three hierarchical layers is discussed below.

All optimizations were constrained to lie below the workstation interface and consisted of implementing HD63484 specific routines to replace the GKS simulation functions that had previously been done slowly by Rutherford FORTRAN subroutines. Preliminary use of RAL GKS showed up the areas where processing was unacceptably slow for use in an interactive application and these provided the target for optimization.

6.5.1 The Fortran Layer.

The FORTRAN layer was initially based on the TEKTRONIX workstation driver supplied by Rutherford. The optimizations caused it to diminish in size as more and more functions migrated into the lower two layers of the hierarchy. Inquire functions and set functions were not modified at all as the Hitachi device had no facility for private state list storage, and there were no advantages to be gained by implementing one in the workstation handler.

6.5.2 The HD63484 Handler.

Initially the workstation handler only acted as the interface between the FORTRAN layer and the device driver. As GKS was put into use so the system bottlenecks showed up. One particular problem was the handling of segments. Since a workstation should be able to store its own segments, a set of segment storage and manipulation routines was written for the handler. These routines replaced the segment storage facility previously managed by the GKS front end routines.

The segmentation functions provided for RAL GKS use a Central Segment Store(CSS) to hold segments for workstations that do not provide their own segment store, but, when used, the CSS was found to be too slow for interactive applications. A workstation specific segment store has been implemented to overcome this problem. When parameters are passed from the FORTRAN section of the Hitachi workstation driver, they must first be formatted to provide the appropriate character stream for the UNIX HD63484 driver. These character streams are packed into vectors which are in the correct form to be passed directly to the HD63484 write routine. If segment storage specific to the HD63484 is required, it is most convenient keep the data in this packed form. When deleting a segment, the whole picture, excluding the deleted segment can be re-drawn by scanning through the segment store and passing the packed

primitive vectors directly to the Hitachi driver. In the event, this method was much faster that the CSS but still too slow for rapid segment manipulation involving area filling. A graphical technique was used to show picture changes when segments were deleted which involved making a note of the background colour and using this to simply re-draw the deleted segment. Although the picture cannot be completely restored using this method, it does provide a good approximation if a considerable amount of the screen area is in the background colour. An all encompassing redraw of the image can be done using the "REDRAW ALL SEGMENTS" function of GKS during a non speed critical section of the program. Other redrawing techniques such as using the EXCLUSIVE-OR operator on the data held in the frame store were considered, but this was thought to cause a more distracting effect than the chosen method.

The Hitachi segment store is constructed as a double linked list of segment pointers. There is a segment pointer for each different segment that has been created. A segment pointer contains information about the segment name, its visibility and its priority. It also contains a pointer to a linked list of the primitives that comprise the segment. Two segment pointers are declared so that their scope makes them continually available to all the segment and primitive management routines. They point to the start of the segment chain and the current segment respectively. Figure 6.1 shows how this structure is organised. The C structure typing is used to generate the general purpose types to build this structure. The actual primitive records that are held within a segment are the vectors that get passed to the HD63484 driver. Each vector has appended to it a header, which consists of a pointer to the next primitive vector and a size parameter giving the number of bytes to be transferred to the HD63484 driver. The clipping rectangle is also stored in the primitive vector, together with the respective primitive attributes, so that the clipping rectangle

and attributes of a primitive are bound to it correctly. The functions supported by this segment store include open, close, rename, delete, and set segment visibility.

As well as controlling output functionality, the handler was used to control the input of data from the BBC computer. The tasks to be managed include initialization, echoing of input values when appropriate, and returning values to the FORTRAN layer when the operator signals completion. Several cursor types were supported, including the cross hair, rubber band and rubber box. The cross hair cursor uses the hardware features of the HD63484 and the others use the exclusive-or drawing mode of the HD63484 to highlight the cursor position and then hide it again when it moves to another position.

The hardware character set is programmed by the handler during the initialization stage of the device during the call to OPEN WORKSTATION. Other fonts can be implemented, and loaded using the font setting command.

6.5.3 The HD63484 UNIX Device Driver.

The high level command interface of the HD63484 has allowed the UNIX driver for this device to reflect the graphical functionality of the GKS primitives without becoming too large. If a uniform interface of this type can be achieved, then the tasks to be performed by the handler will be made more straight forward. An example of this is the way that segments may be stored, ready for passing to the driver. The interface specification is given in appendix H. Currently its functionality reflects the primitives polyline, polymarker, text and fill area. The clipping rectangle and attributes are bound to each primitive so that they may also be passed to the driver. Clipping is performed automatically by the HD63484 but other functions require some simulation to

be used. The primitives for polymarker and fill area for example, must be constructed from lower level functions. Hardware facilities are exploited to manage attribute simulation in an efficient manner.

A UNIX, special character device has been created within the filing system called "/dev/hitachi". The device has read and write status, but only the write entry in the driver is used. The read routine could be used to enable the raster operations INQUIRE PIXEL to be implemented at a later date. The first two bytes of the command packet are copied from the user's data area into the kernel where they are compared with the command options supported by the driver. If a succesful match is found then the required action will be called. If not, an input/output error is flagged to the user. Before data is copied into the kernel from the user's space, a test is made to ensure that the required amount of data has been made available from the user by checking the appropriate kernel byte count parameter which was set from the user's arguments during a read or write by the kernel. If this test fails an error is reported.

If a valid request has been made, the service routine is entered, where a function specific amount of initialization data is copied into the kernel. This data includes the attributes and clipping rectangle of the primitive, and the number of coordinate points associated with the primitive. This data is used to initialize the HD63484 or prime a simulation algorithm ready for drawing. Finally, the positional data is copied into the kernel ready to be fed directly to the HD63484 or to a drawing algorithm such as area fill. There is no limit on the number of positional parameters that may be passed to the driver and no double buffering is necessary.

Each primitive function will be examined in turn to describe how the features of the HD63484 were exploited, or how simulation algorithms were

implemented.

The initialize driver function is called as a consequence of an OPEN WORKSTATION on the Hitachi system. It initializes the timing parameters of the HD63484 to suit the display monitor, in this instance, the timing is suitable for the Mitsubishi C6920[46]. It also selects the display resolution of 1024x1024 pixels with 8 bits per pixel which is the optimum configuration for the system as described in chapter 4. Two colour numbers are also initialized, colour zero being black, and colour one being white.

Currently, the polyline attributes of colour and linetype are handled within the driver. Colour, as in all primitives, is handled directly by the HD63484 by entering the colour number into its colour register. Linetype simulation is performed by the HD63484 using the data in its pattern RAM to modulate the intensity of the line as it is being drawn. The five styles specified in the GKS documentation are supported. Figure 6.2 is a photograph taken from the monitor screen to show a variety of linetypes and line colours.

The polymarker function can simulate all the attributes of type, scale and colour. The five marker types specified in the GKS standard are generated by programming their bit patterns into the pattern RAM of the HD63484 which then draws that map into the frame store at the required position. The bit maps are stored within the driver and are not user programmable, although this feature causes no problems when the driver is used with GKS. Marker scale factors are effected by the zoom drawing function of the HD63484, allowing factors from 1 to 16 to be used. Figure 6.3 is a photograph showing several sets of polymarkers.

The attributes associated with text are used to simplify the handler

functions. The text font can only simulate CHAR and STRING precision, although the font in this instance is programmable. An 8x8 grid can be used to define new fonts, which are used to program a font table within the driver. A font programming function accepts the character grids from the handler and moves them into the kernel data space where they may be accessed more quickly than if they were passed from the handler every time that text was being plotted. Characters are drawn using the same method as markers, with their size attribute being simulated by the HD63484 zoom drawing function. A limited accuracy of text rotation is supported by the HD63484. Width and height expansion factors are simulated. A character string can be passed to the driver together with the incremental plotting data which will be used to position the string on the screen. The stroke precision simulation (supplied with the Rutherford-ICL GKS) is still available, and is implemented using the driver polyline function. Figure 6.4 shows several text string plotted using CHAR precision while figure 6.5 shows STROKE precision text.

Area filling required more simulation to be done by the microprocessor. The F77 routines supplied by RAL were slow and worked in floating point format. These general purpose routines were replaced by a HD63484 specific area filling algorithm. The HD63484 supports boundary filling, and, when the device was first released it was planned that the HD63484 boundary fill could be used for non-intersecting polygons, and a conventional scan conversion algorithm[13] used for self intersecting ones. A specification update on the HD63484 tightened the operating conditions for boundary filling to an extent where it could not be used to improve the performance of GKS area filling. The problem of detecting intersecting polygons may have resulted in more processing than a conventional filling algorithm anyway. Instead, a fill algorithm was used for all non-rectangular shapes, and the HD63484 rectangle fill function used otherwise.

The algorithm was written in C, initially as a user program that accessed the HD63484 using the general read and write utilities that had been built into the driver. All computation was in integer format for efficiency of operation and permitted the debugged filling routines to be incorperated in the HD63484 driver at a later stage. The HD63484 clipping function removed the need to implement a polygon clipping algorithm. In operation, the algorithm fill is as fast as the HD63484 when boundary filling, but it does prevent the processor from performing other tasks during that time. Figures 6.6 and 6.7 show photographs of a self intersecting polygon that has been using the above algorithm. In figure 6.7, the HD63484 clipping mechanism has been enable.

6.6 Conclusions.

The powerful graphical processor of the HD63484 has enabled a UNIX device driver to be written with a functional interface that is very uniform in its handling of the GKS abstract output primitives. This has enabled a very efficient segment handling utility to be written so that very little processing is needed to retrieve the segments from the Workstation Dependent Segment Storage(WDSS) and present them in a suitable form to the driver. Input class simulation on the BBC terminal computer also supports a very uniform functional interface which has simplified the workstation handler and consequently improved efficiency. Since simulated input devices in non graphical classes may be operated without the conventional GKS echoing mechanism, complete decentralized operation is possible, further reducing processing in the host computer. The optimizations gained in this way have enabled RAL GKS to be used for interactive applications, whereas before it was too slow when run on the SBC workstation system. The power of the HD63484 has enabled compact primitive simulation routines to be written and

incorporated in the device driver. The close coupling of the system processor and graphics processor has further improved performance over workstations that use serially connected display devices such as the Tektronix terminal that was first used to test RAL GKS.

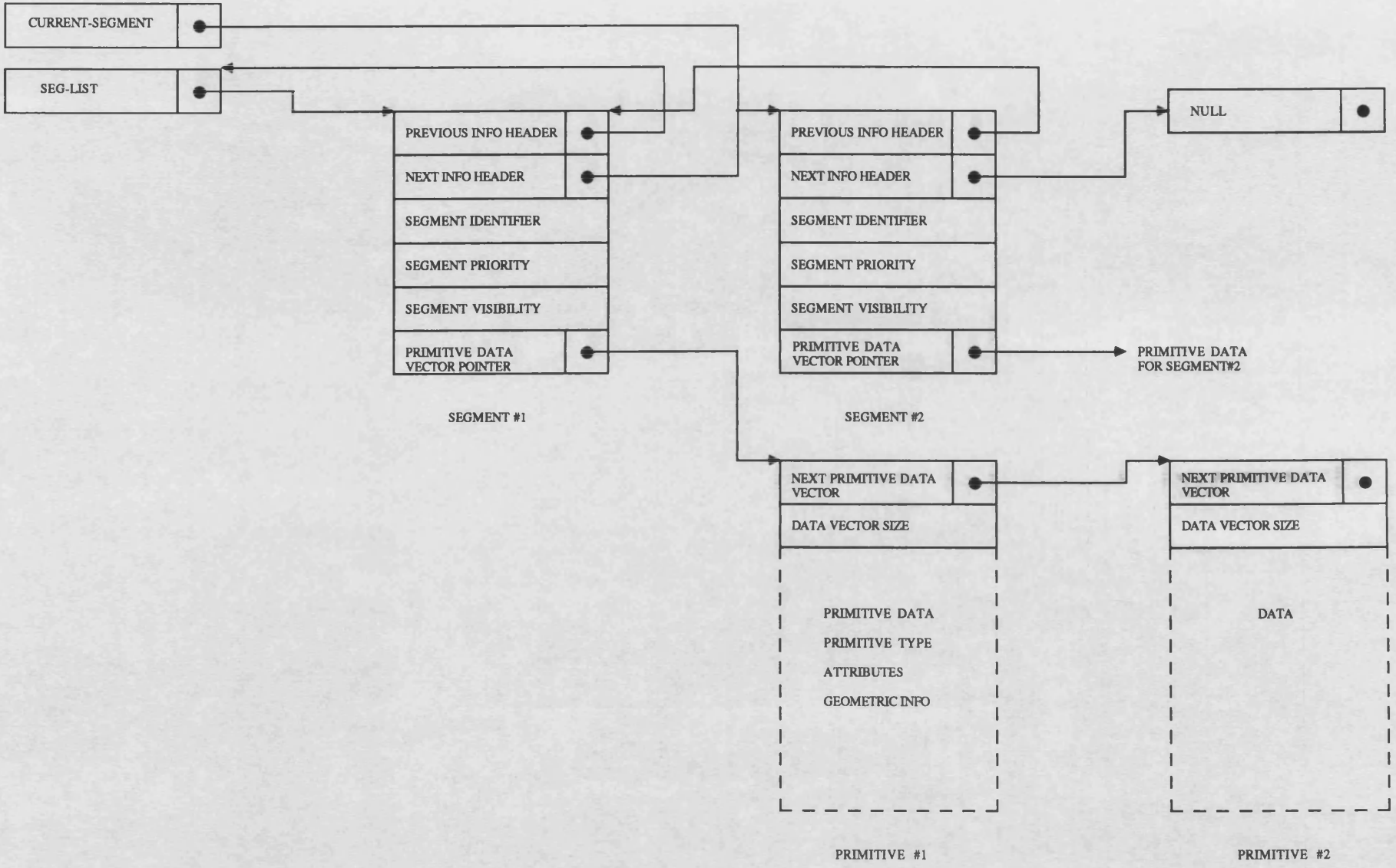


FIGURE 6.1 WDSS FOR THE HITACHI HD63484 GRAPHICS SYSTEM

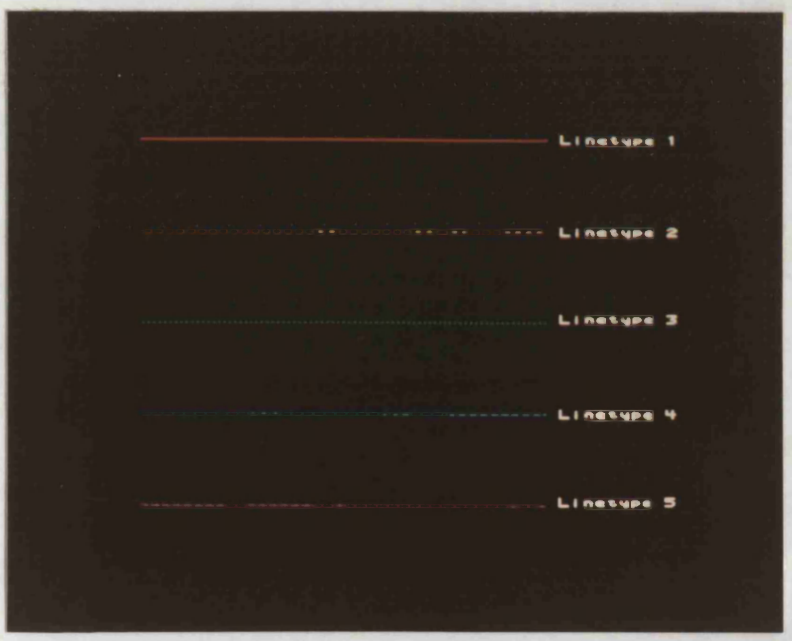


FIGURE 6.2 SIMULATED LINETYPES AND COLOURS AS OUTPUT USING THE GRAPHICAL KERNEL SYSTEM

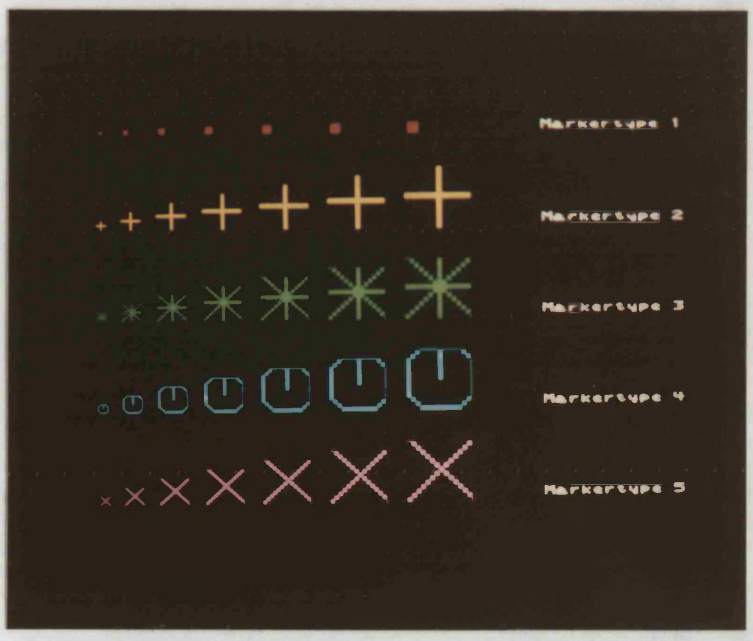


FIGURE 6.3 SIMULATED MARKER TYPES AND COLOURS AS OUTPUT USING THE

THE GRAPHICAL KERNEL SYSTEM

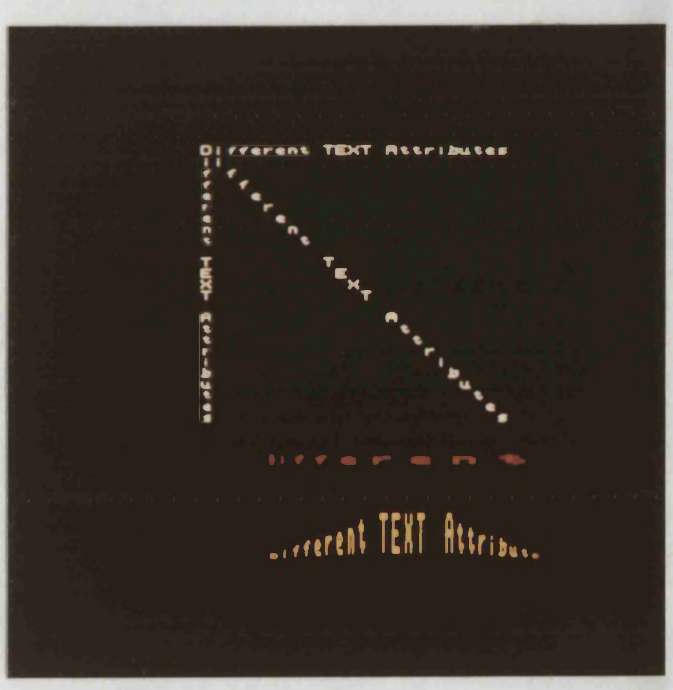


FIGURE 6.4 SIMULATED CHARACTER PRECISION TEXT AS OUTPUT USING THE GRAPHICAL KERNEL SYSTEM

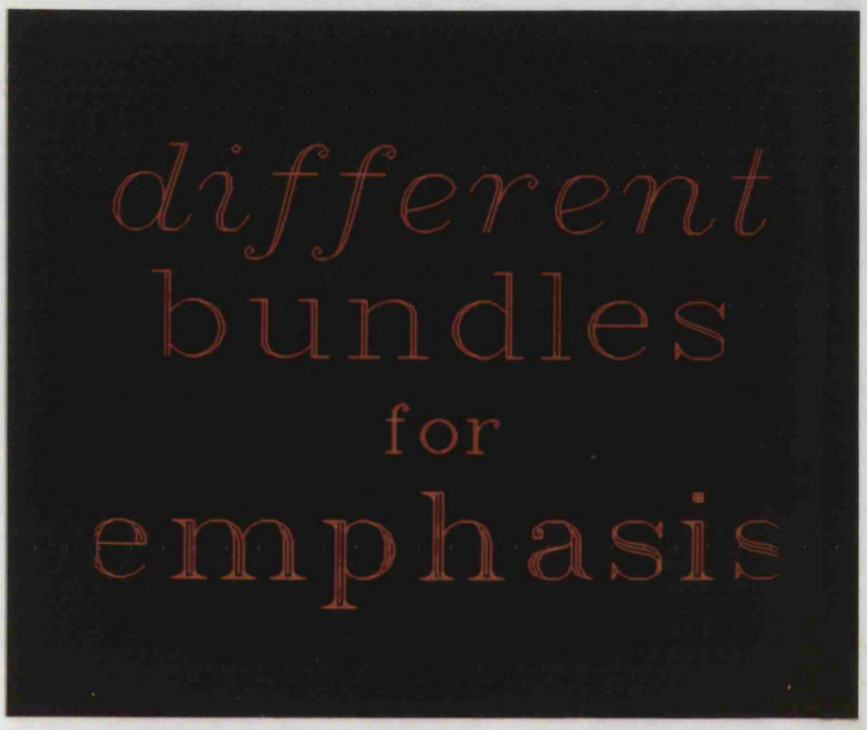


FIGURE 6.5 STROKE PRECISION TEXT AS GENERATED BY GKS

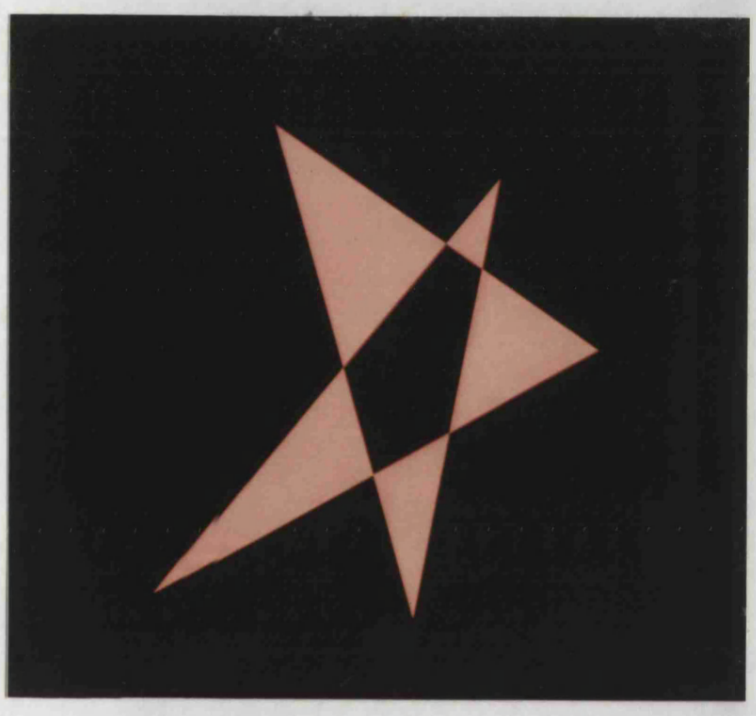


FIGURE 6.6 AREA FILLED POLYGON AS OUTPUT FROM GKS

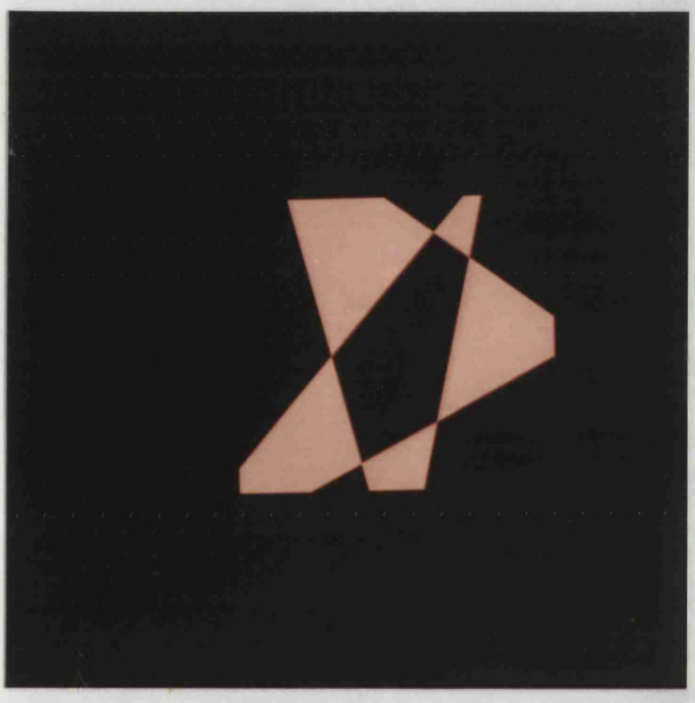


FIGURE 6.7 AREA FILLED POLYGON WITH HARDWARE CONTROLLED CLIPPING

AS OUTPUT FROM GKS

CHAPTER 7

COMPUTER AIDED DESIGN ON THE SBC WORKSTATION

7.1 Introduction.

This chapter brings the development of the SBC workstation to its logical conclusion by describing the final stages of system integration and testing. By way of a summary, the following hardware and software components were integrated to form the complete SBC workstation. The first of these was the main processing system which consisted of the Single Board Computer with a 19 inch backplane, power supplies and a hard disc system, all of which were housed in a single desk top unit. The HD63484 graphics system, a 2 Mbyte memory card and a Multilink networking card were also plugged into the SBC backplane within this unit. A BBC Computer and VDU were interfaced via a standard RS232 link to the SBC workstation and configured to run the extended terminal/input device emulation program. The joystick and mouse devices were connected directly to input ports on the BBC computer. A Gould pen plotter[52] was connected to the second RS232 port of the SBC workstation. The addition of a high resolution colour monitor for the HD63484 graphics system completed the system integration. A photograph of the complete system is shown in figure 7.1.

The software development environment for the SBC workstation was supported by the UNIX version 5.2 operating system while the Rutherford/ICL GKS front end provided the application interface to the workstation graphical devices. The drivers that had been written for the HD63484 graphics system and the Gould pen plotter were interfaced to the GKS front end.

The main design criteria for the SBC workstation can be summarised by the mutually exclusive requirements of performance, generality, standardisation and cost. The design of the SBC workstation required careful consideration on each of these points, and the resulting system was analysed to discover how these constraints had affected its capability in handling large CAD tasks.

The performance of the SBC workstation was assessed by running a large CAD program that was both memory and processor intensive. This program also required fast picture manipulation, and hence the performance of the graphics system was also tested. A comparison in performance was also made with the computing environment under which the CAD program had originally been developed. This was particularly appropriate, since the host development system was a mainframe computer with satellite graphics terminals.

Testing the SBC workstation for standardisation was automatically achieved by porting the above CAD program (which had been written using the GKS applications interface) onto it. The ease with which the porting task was achieved was attributed to the adoption of the GKS interface to the graphical hardware, and the use of the Fortran77 programming language. This latter point had already been proven by the ease with which the Rutherford/ICL GKS front end had been ported onto the SBC Workstation.

The generality of the SBC Workstation stems from the use of the UNIX operating system to support program development and execution. Unix has emerged as a de-facto industry standard and can hence provide a familiar interface to a large proportion of computer users. Additionally, UNIX supports a good set of tools for program development and debugging. Compilers are available for Fortran and C, and word processing facilities enable documentation to be prepared, all on a single machine.

The cost of the SBC workstation is kept to a minimum by employing VLSI devices which offer a maximal functionality to cost performance. The power of these devices must be sufficient to handle processor intensive CAD applications, particularly when baring the overhead associated with supporting standard device independent interfaces such as GKS.

Other features of the SBC workstation not usually found on other commercial equipment were also tested. For example, the ergonomics of the SBC Workstation was assessed to determine how effective the splitting of GKS display surface from the general purpose alpha display affected interaction. The graphical input system is also tested to assess the mixed use of keyboard, mouse and joystick input devices.

Most of the performance tests were subjective, being made either by myself, or the original author of the CAD application program. The results are presented using photographs taken directly from the display screens and also by hard copy prints produced by the pen plotter.

7.2 Computer Aided Design on the SBC Workstation.

Through the addition of graphics hardware and software, the Single Board Computer system was converted into an interactive workstation that was suitable for a wide range of CAD applications. The key to successfully achieving a productive working environment for CAD hinges on the development of an efficient Man Machine Interface(MMI). The operator should feel that he is interacting, not with with the computer hardware, but rather, at a much higher level with the actual design model[62]. Considerable research effort is still being spent on all aspects of CAD, and this may result in better design methodologies for the MMI.

Currently, there are several popular techniques used when designing an MMI. Some are based on question and answer schemes, some on multi-level menu driven schemes while others employ a command language for the man/machine dialogue[63]. The SBC workstation is well equipped to support the human interface of the MMI. Its input devices, coupled with the high resolution graphics display and the alpha input terminal offer comprehensive facilities via the GKS primitives to support today's interactive CAD applications. The scope of activities to which CAD is now applied are vast, and hence an exhaustive evaluation of the SBC workstation to all CAD applications is not possible. However, a good measure of the performance and suitability of the SBC workstation to handling real applications may be made by running a comprehensive CAD package that involves considerable user interaction and generates large processing loads.

7.3 Molecular Modelling by Computer.

One discipline that can benefit from the application of CAD to its problem solving is molecular modelling. The package chosen to demonstrate and test the SBC workstation was a molecular modelling suite of programs collectively known as GLXMOD. This package was kindly provided by Dr Murray Rust from the Glaxo Pharmacutical Research Centre, Greenford, London. The complete package was written in Fortran 77 and used the GKS applications interface to control the graphical input an output devices.

A brief overview of molecular modelling is given to highlight the subject areas where a computer may aid problem solving in this discipline. It also gives an indication of the computer resources that are typically required by molecular modelling programs.

Chemists and pharmacists who are concerned with the analysis of molecular structures have, in the past, used stick and ball models to help them visualise these structures in three dimensions. The shape and dimension of a molecule's surface is an important criteria in many different applications. A model of the molecular surface may also be constructed using plastic spheres to represent the constituent atoms. These spheres are coloured to represent atom type, and their radii represent the Van der Walls radii of the atoms. The spheres slot together to produce what is known as the CPK space filling model[64] of the molecule.

Building stick and ball or CPK space filling models showing atom position and bonding features are inflexible, time consuming and prone to error. In some instances, the molecular dimensions and atom positions must be obtained from reference books. Large 3D models may become cumbersome and are prone to disfigurement. Computers have been used to solve many of the problems associated with modelling molecules with mechanical structures, and the use of interactive computer graphics has enabled manipulation of the generated images which can give the operator as much information as a conventional mechanical model. One of the earliest papers on molecular modelling by computer was published by C. Leventhal in 1968[65]. A monochrome monitor was used to display wire frame models which could then be manipulated in real-time. Since that time, considerable effort in both research and commercial establishments has been made to exploit the power of computers and computer graphics to aid users who require viewing and on line analysis of atomic structures. The chemist can use the storage media of the computer to hold the relevant data for many thousands of molecules which can then be quickly retrieved to draw representations of the molecules in either stick or space filled form. The computer may also be used for analysis, providing information such as atomic bond lengths, angles, or performing energy calculations to determine the

relative stability of a particular molecular configuration.

The atomic structure of a protein molecule may be obtained by first crystalizing the protein and then using X-ray diffraction techniques on the crystal. Unfortunately, it is not possible to crystalize certain proteins, and a different method must be used to detect their structure. A protein is constructed from amino acids and, if their type and connectivity are known, it becomes possible to predict the structure of the protein using the properties of these amino acids. The computer may aid in this synthesis by allowing the operator to configure the protein whilst performing energy calculations to show the stability of a particular structure. Hence, an incremental process is possible, which, if guided by the operators experience and expertise, may eventually lead to the correct protein structure.

Molecular modelling is also invaluable in drugs research and design, particularly where the drugs are built from biological molecules. These drugs function by providing an active site on their surface into which the unwanted harmful molecule, known as a substrate, may fit. In the reaction that occurs between a drug and a substrate, the substrate becomes locked onto the drug and its effect is neutralized. Fast viewing and manipulation techniques made possible by powerful computing facilities enable the chemist to quickly identify potentially valuable drugs whose surface shape will be such that the substrate will fit as closely as possible. To aid in this process, a technique known as docking is performed on screen by the computer, in which the substrate is moved up to the molecule under test to show how close the fit may be. The shape of the docking point may be manipulated to optimise the fit and minimize the energy involved when the enzyme and its substrate react. The use of molecular modelling can considerably speed the process of drugs synthesis

by thus identifying likely enzymes and rejecting unsuitable ones, which would in the past have been done by physically producing the molecules and then testing them, possibly using animals. The cost and time savings of molecular modelling in the synthesis and testing stages can therefore be considerable.

Finally, very large molecules may be viewed on a global scale so that particular features may be assessed. One of these interesting areas may then be expanded for closer scrutinisation. With computer graphics this is an easy task and the operator may effectively be able to scan over the molecule, zooming in and out as necessary.

It is clear that molecular modelling by computer brings a requirement for fast picture manipulation and creates large processing loads. If comprehensive molecule data bases are be used, then a large volume permanent storage device will additionally be required.

7.4 The Development Environment for GLXMOD.

At the Glaxo Pharmacuticles Research Centre, the development environment for molecular modelling was based on a single host computer with satellite alphanumeric and graphics terminals connected to it. This was a classic configuration for multi-user time sharing computing, and hence offered a good oppertunity to compare the performance of a specialized CAD application running in a time sharing environment with that of a microprocessor based single user workstation.

The host computer that was used to develop GLXMOD was a DEC VAX machine running the VMS operating system[66]. VMS is a virtual memory operating system, and hence it was possible to run very large programs that operated on large data structures. The available processing power in a time

shared system is obviously linked to the number of simultanious users, and hence, under extreme loading, the response time will be insufficient. This unpredictability in performance is an obvious drawback to running CAD applications on a multi-user system.

Both raster and vector scanning graphics terminals were connected to the VAX computer. The vector scanning devices were very specialized, having their own local real-time display processors which were connected via a parallel link to the host. A performance comparison between a system of this type and the SBC Workstation was unrealistic purely on the grounds of cost. The raster scan graphics terminals were connected to the host computer via serial RS232 lines running at 9600 baud and some of these employed local processing power to boost performance.

GLXMOD was developed on a Sigmex 6200 raster scan graphics terminal[50] which had a resolution of 720 by 512 pixels with a maximum of 256 simultaniously displayable colours. The colour palletting allowed each of these colours to be specified using 256 level quantisation for the primary colours. This terminal had a functional interface modelled on the GKS workstation function set. Its serial communication link with the host computer could be thought of as the logical data link through which GKS communicates with its workstations. Ideally, a full implementation of GKS on this system would have required a GKS front end simulation to be run on the VAX computer.

In practice, a set of subroutines was written to map the workstation interface functions directly onto their respective GKS functions. These functions were named according to the standard GKS language binding. As a consequence of this mapping, certain features of the GLAXO system did not comply fully with those required by the GKS specification. For example, only a single

transformation from world to device coordinates was available, since this was being mapped onto the underlying unique workstation transformation. The problem with this approach was the lack of picture composition capability which would normally be achieved by the definition of multiple normalization transforms.

Because the Sigmex 6200 terminal was capable of GKS workstation emulation, considerable graphics processing was offloaded from the host processor. Functions in the viewing pipeline for clipping and workstation transformation were provided, together with the direct emulation of the GKS primitives. Storage for segments was local to the Sigmex terminal and hence the burden of segment manipulation was also offloaded from the host processor. The use of intelligent terminals also helps prevent the serial communications link from becoming a bottleneck by helping to reduce the quantity of data flowing. The Sigmex terminal also supported two physical input devices, these being a mouse and joystick control, both of which could be operated as logical GKS input devices.

As well as the fundamental differences between the GLAXO and SBC systems, the ergonomics of the two systems were also different. The partitioning of the GKS workstation display and general purpose console terminal into physically separate pieces of hardware allowed different and more flexible styles of interaction to be designed. Because the data link between processor and graphics system on the SBC workstation was much faster than the host development system, the screen update rates had the potential of being faster. The resolution of the SBC Workstation was, however, higher than the SIGMEX terminal and hence more graphics processing on the bit map display was required by the workstation.

The microprocessor based SBC workstation must be powerful enough to support applications that expect the support given by a mainframe/intelligent terminal configuration. To achieve this, the processor intensive areas were analysed to identify where VLSI solutions could be employed to provide a multi-processing system while keeping costs to a minimum. For the SBC workstation, this has been achieved by employing an intelligent CRT controller and using the intelligence of the BBC computer terminal for input processing. The potential of the SBC workstation at competing on terms of performance with the GLAXO VAX based system is evaluated, considering the low cost approach taken to achieve the system functionality of the SBC workstation.

7.5 An Overview of GLXMOD.

The molecular modelling package supplied by Dr Murray-Rust was named GLXMOD and comprised of a large number of subroutines written in FORTRAN77 and a data base of molecular information stored in files using the CHEMGRAF format[67].

GLXMOD is a menu driven interactive molecular modelling program which provides functions for viewing and synthesis with the option to store any new or modified molecules on permanent storage for re-use at a later time. The path through the menu is organised as a two level tree structure, with some functions being multiply accessible across menu pages. Each menu contains up to a maximum of 24 options.

Since GLXMOD uses the GKS primitives to obtain graphical input and output, its potential for portability to the SBC Workstation was high, and this proved to be the case in practice. The hierachical structure of GLXMOD is very well defined, being partly imposed by the structure of the menu system. At a lower

level, program modules may be grouped by their functionality into the class headings of maths utilities, GKS utilities, and menu utilities.

The most fundamental data structure in GLXMOD holds the molecule structure. To do so requires for a three dimensional position, a valence, an atom type, and connectivity information, for each atom of the molecule. To display a complex molecule will therefore require large data structures and for a 500 atom display, almost 1Mbyte of memory is required. When GLXMOD was first compiled, the information pertaining to a maximum of only 20 atoms could be held in store at any one time. At GLAXO, the maximum number of atoms had been set to 500, but when this figure was included in the compilation on the SBC workstation, the subsequent core image generated for GLXMOD was greater than the UNIX imposed maximum process size of 1 MByte. To overcome this problem, the UNIX kernel was rebuilt so that process images could extend to 2Mbytes. Since the SBC workstation was furnished with 3 Mbytes of memory, this extension was easily accommodated.

Data was transferred between the "in memory" molecular data base and the secondary storage by read and write menu functions. The only modifications to GLXMOD which were attributed directly to the differences between the host development system and the SBC workstation were to the file name specifications contained in the Fortran "OPEN" statements for file I/O. These differences were however very easy to isolate and hence correct for.

Because the Sigmex 6200 terminal was connected via a serial link to the VAX computer, data buffering in the host was employed to improve data transfer efficiency. The GKS deferral mode was toggled within GLXMOD to achieve the required response, but when ported to the SBC workstation, buffering was not required, and all actions were performed using the deferral state "As Soon As

Possible".

7.6 Techniques for User Interaction with GKS.

GKS offers many powerful features which should enable the straight forward construction of device independent man machine interfaces. Porting GLXMOD to the SBC workstation tested this device independence in the light of differing hardware configurations and differing system performance due to the different task partitions in the overall processing associated with the application. Certain features of the SBC workstation did enable GLXMOD to be altered slightly to improve efficiency. This section highlights the areas where these opportunities arose, and also where the GKS programming techniques were optimum for both systems.

7.6.1 Dynamic Picture Manipulation

Animation was achieved by using two segments and toggelling their visibility attributes. The animation process consisted of drawing primitives to the invisible segment whilst the visible one was being viewed by the operator. The image update was achieved by first making the visible segment invisible and then making the invisible segment visible. This choice of animation style was particularly suited to the segment handling methodology used for the SBC workstation. To maintain the representation of the picture in the operator's mind, it is necessary to cause updates to occur as quickly as possible. It is undesirable that the image should be constructed so slowly that a noticeable delay be introduced in between plotting each display primitive. When rotating or manipulating molecular structures, considerable floating point arithmetic must be performed by the application program and also in the viewing pipeline of GKS. If the segment is updated when it is invisible, the delays due to these

calculations will not be seen by the operator. The segment store for the SBC workstation is optimised to suit data transfer directly to the graphics processor, and no intermediate calculations are necessary. Hence, when the segment is made visible, the segment store can feed the image already stored in device coordinates directly to the graphics processor such that a continual drawing action is maintained and the picture is updated in an uninterruped fashion. In this case the segment store was performing as a primitive buffer which helped improve the animation effect. Using the segment store in this manner does decrease the potential maximum picture updates per second over the simpler method of delete and redraw, but even so, the benefits of smooth animation gained by segment visibility toggleing outweighs the use of the simple method. When the invisible new image is being drawn, the old segment remains visible to give a stable image for the operator. The visible segment is then made invisible, which is fast in the SBC workstation because primitives are simply redrawn in the background colour, and then the new image is made visible by toggleing its segments visibility.

7.6.2 User Prompts.

In this context, prompts are taken to mean text strings that are used to relay information to the user, and in GLXMOD, these prompts serve several different functions. Firstly, the GLXMOD menu options must be displayed so that the operator may select his required action. Secondly, prompts must appear to either warn the operator or guide his actions during the course of interaction with the application program. Finally, prompts may report the results of calculations to the user.

Two methods may be used to represent this textual information to the operator. In the first instance the display is capable of producing both the text

and the graphical output, the former being produced by the standard input/output functions of the operating system whilst the latter is produced by the application program using the GKS primitives. In some instances, the complete display surface is split into windows therefore enabling simultaneous display of ordinary character strings and graphically produced information on logically separate areas of the display. The Sigmex 6200 terminal used at Glaxo employed this method, but the alphanumeric and graphical screens both occupied the whole of the display surface. The alphanumeric text appeared to overlay the graphics window, and its visibility could be toggled to make it invisible if not required. The SBC workstation however adopted the second approach which required two physically separate display devices for the alphanumeric and graphics screens. This approach allowed a more flexible method of interaction to be used, particularly since the alphanumeric terminal was also used as a GKS logical input device emulator.

When GLXMOD was ported onto the SBC workstation, menu choice selection was achieved by using the GKS text primitive to plot an appropriate list of available options down the right hand side of the display. Figure 7.2 shows this menu layout on the screen. To pick an option from the menu, it was necessary to use a locator input device which returned a Y coordinate value which could, by comparison, be associated with one of the options. There were problems associated with this approach, since the menu and the molecular image are sharing the same display device and are hence sharing graphical bandwidth which slows overall performance. The generation of text from GKS is not particularly efficient since it must use a floating point representation, and there are many attributes which may potentially require time consuming simulation. As an extreme example, only stroke precision text was available when GLXMOD was first running on the SBC workstation and its simulation from

many font strokes using line drawing was unacceptably slow.

The selection of menu options was best modelled using a GKS choice input device, and hence this approach was used instead of the Locator method for picking menu options originally employed in GLXMOD. The choice emulation described for the BBC terminal was used, and the menu therefore appeared on the alpha screen with all interaction being handled within the BBC computer, which thus removed the need to have the menu display interfering with the graphics image. The value returned by this choice device directly represented the actual choice number, and hence no additional processing was required by the SBC workstation. In use, the BBC choice emulation proved very successful and improved the MMI. Figure 7.3 shows the display produced on the alpha screen by some of the different menu level option pages.

There were two methods of relaying instructions to the operator that were used by GLXMOD to prompt him for a particular response. When interaction was closely related to other operations being performed on the graphics screen, prompt strings were displayed in highlighted boxes at the top of the screen. Figure 7.4 shows a prompt of this style, the overhead of which was was sufficiently small since the amount of text could usually be plotted without delay. For warning messages, GLXMOD used a different method which resulted in flashing the text on the screen for a short period of time. This involved making the alpha overlay plane visible and plotting the text string upon it. For the SBC workstation, this was not possible, since the implementation of an overlay plane would have halved the number of simultaniously displayable colours. Instead the alpha screen was used to display the warning messages, which would remain visible until further messages or other text caused it to scroll out of the visible text window. When input emulation on the BBC

computer was in progress, the appearance of warning messages caused no problem as there was always a least 9 text lines available at the bottom of the display for the alpha window.

7.7 Running GLXMOD on the SBC Workstation.

Having discussed the features of GLXMOD in its development environment and examined how its functionality would map onto that of the SBC workstation, the next section describes some of the features of GLXMOD running on the SBC workstation. When GLXMOD was entered, the top level menu was displayed on the alpha screen, and this is shown in the photograph in figure 7.3. The options available from this menu include a depth cueing function, geometric attribute calculations such as atom separation, a bestview function and functions to toggle various attributes of the display such as label visibility and hydrogen atom visibility. A hardcopy function was also available, which outputted the currently displayed molecule to the pen plotter. The sequence of photographs in figure 7.5 illustrates the effects of some of the functions that can be performed at this level. The Depth Cueing function modulates the intensity of the bond and its atom, dependent upon the distance of that atom from the viewer. The SBC workstation was well suited to this application due to its 256 level quantization capability for each primary colour. Three options were available to calculate the distance, angle and torsion angle between groups of atoms. This operation was very easy to implement since it only required information to be extracted from the molecule data base that was held in memory so that calculations returning the required values could be made. The Bestview option projected the molecule onto its inertial plane, such that this plane was that of the workstation display surface. The atom labelling could be turned on or off, this feature being particularly valuable when large

molecules were being displayed, since the labels tend to hide detail in those circumstances. The hydrogen atoms of a displayed molecule may be made invisible, this feature again improving legibility for large molecules. The last feature implemented at this level was the Hardcopy function. In the GLAXO implementation, special routines were used to access the "in memory" molecular data base and plot the molecule onto the hard copy device, and these routines were neccessarily device specific, since the GKS functionality was distributed to the external graphics devices. The hard copy device at GLAXO did not have a GKS interface, therefore special routines were required to drive it. In the SBC workstation, this situation did not arise. The hard copy device had its own workstation driver, and could be opened, activated and used in exactly the same way as the raster scan workstation. Hence, the same code could be used to drive the hard copy plotter as the raster system. Figure 7.6 show the typical output from the Hardcopy facility.

Some entries on the top level menu indicate a transition to a lower level menu where more specialzied functions were available. The three sub menu entries which could be selected from this level are Build, Manipulate and Spacefill. The version of GLXMOD supplied by Dr Murray Rust did not include any of the Spacefill sub menu options. The second level Build menu display is shown in figure 7.3. In particular, the sequence of photographs in figure 7.7 illustrate how the Build function was used. In step 3 of this sequence the menu of possible atoms which could be used as building elements is shown as it was displayed on the graphics screen. This could of course have been implemented using the Choice input emulation on the BBC computer, but in this instance it was left as a Locator type input selection as this did not significantly degrade performance.

The other options available from the build menu include Join/Unjoin, Add

Hydrogens, Join-All and Hydrogen Join. The Join/Unjoin operation is performed by first identifying the two atoms that are to have their connectivity inverted. The bond between these atoms is then either made, or broken depending upon whether the two atoms were previously joined or not. If the result of breaking a bond is to generate two molecule fragments, then a change of colour will occur in one of the fragments in order that it may be distinguished from the other fragment. The sequence of photographs in figure 7.8 illustrate the decomposition of a complex molecule into smaller fragments and shows the various colours used to represent these fragments.

The Join-All option is a crude hybridization routine which functions by testing inter-fragment atom separations and performing energy calculations to select the most stable joined configuration. Hence, more complex molecules may be constructed from simple collections of molecule fragments. This function was tested by dismantling a large molecule using the Join/Unjoin function and then using the Join-All function to restore all the previously broken bonds of the original molecule. Hydrogen join will connect two fragments together through user nominated hydrogen atoms.

The Manipulate sub menu is shown in figure 7.3. Of the functions provided at this level, Rotate is the most effective for evaluating the performance of the SBC workstation since it involves the processing of many floating point values and also requires efficient segment manipulation to achieve smooth animation. Using this function therefore tested the potential response time of the SBC workstation to the full. The joystick input device working in Locator mode was used to control input for the rotation operation. Figure 7.9 and figure 7.10 show snap shots of two different molecules at different points in their rotation cycle. In practice, these molecules were the largest that could be rotated on the SBC

workstation with an acceptable frame update rate. Subjective test by Dr Murray Rust revealed that the rotational performance of the SBC workstation was only slightly below that of the VAX/Sigmex 6200 system.

7.8 Conclusions.

In the past, mainframe computer and graphics terminal combinations have been the only way to achieve the necessary power for interactive applications. The obvious problem with this approach is the cost of such a configuration and the highly variable processing loads which are generated by CAD programs.

The results presented in this chapter have revealed that the SBC workstation is capable of competing with these more conventional mainframe based systems on terms of performance, whilst representing a considerable monetary saving, to the point where it becomes possible for each time sharing operator of a mainframe system to have his own dedicated workstation. The advantages of reliability and zero loss of performance under extreme system loading are clear. Networking facilities are provided to ensure that communication between systems is possible, and, if needed, users could share a large data base of information. For example, GLXMOD could use a library of molecular information stored on a central file serving reference machine.

Device independency and portability through the use of the Fortran 77 programming language and the Graphical Kernel System has also been proven by the ease of moving GLXMOD onto the SBC workstation. Where changes were made to GLXMOD, these were made to improve its performance on the SBC workstation. These changes were not device specific, but instead they used the GKS facilities that were most appropriate for the dual screen SBC workstation.

FIGURE 7.1 THE COMPLETE SBC WORKSTATION CONFIGURATION

FIGURE 7.2 THE ORIGINAL GLXMOD MENU SCREEN

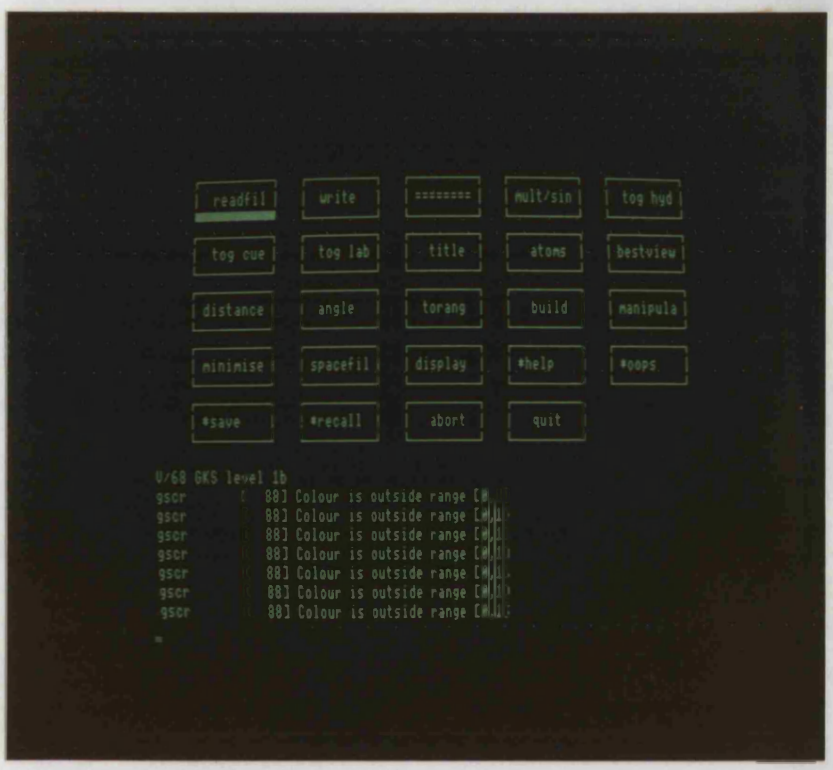


FIGURE 7.3 MENU CHOICE DISPLAY AS SIMULATED ON THE BBC COMPUTER

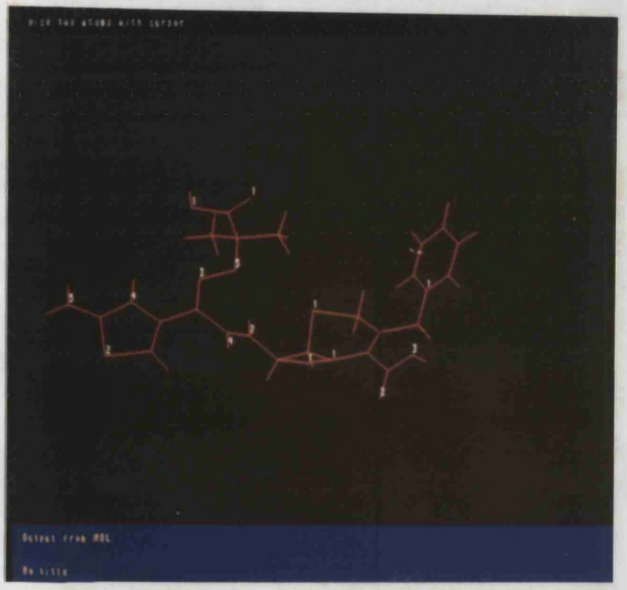
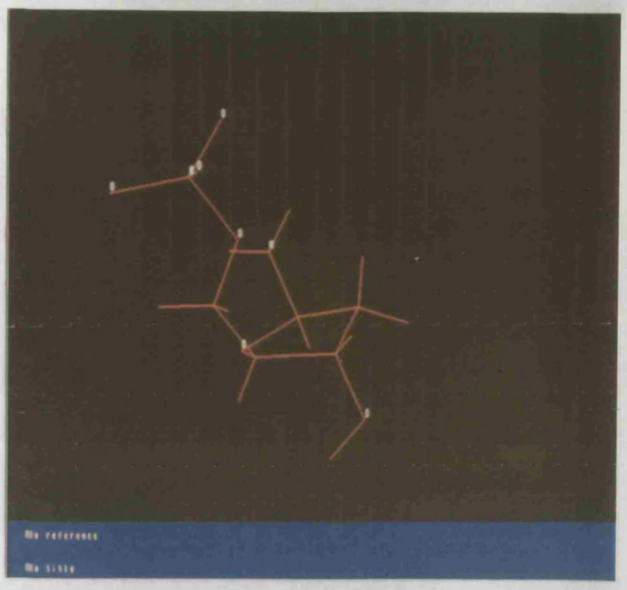
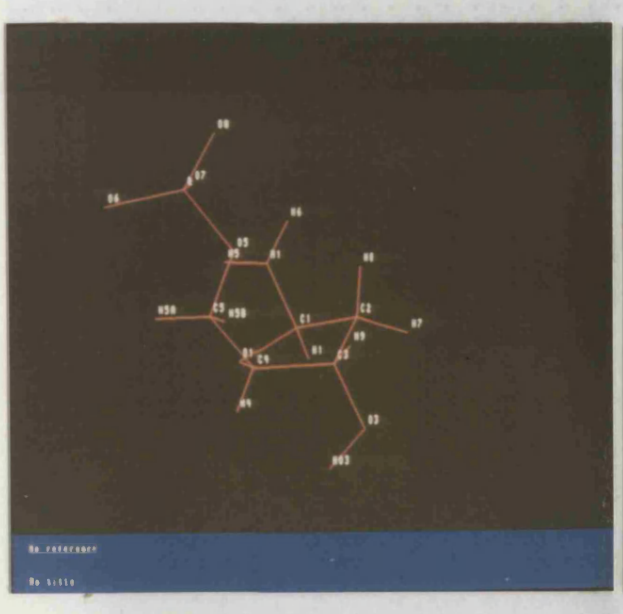
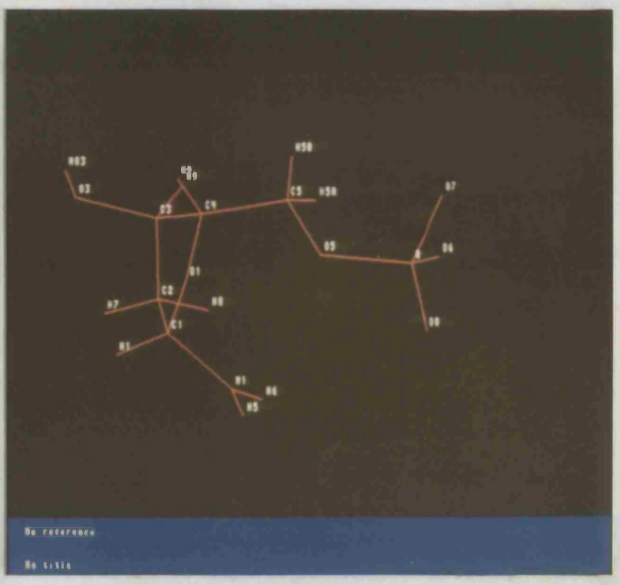


FIGURE 7.4 GRAPHICS DISPLAY PROMPT STRING



MOLECULE AS READ FROM DATA FILE

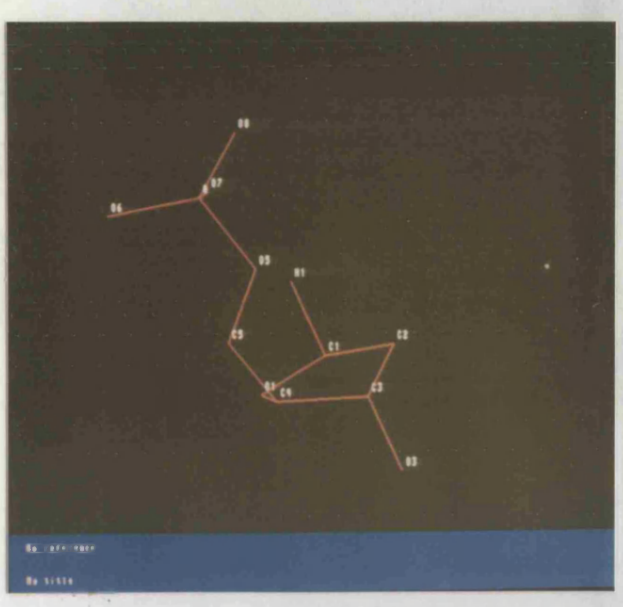




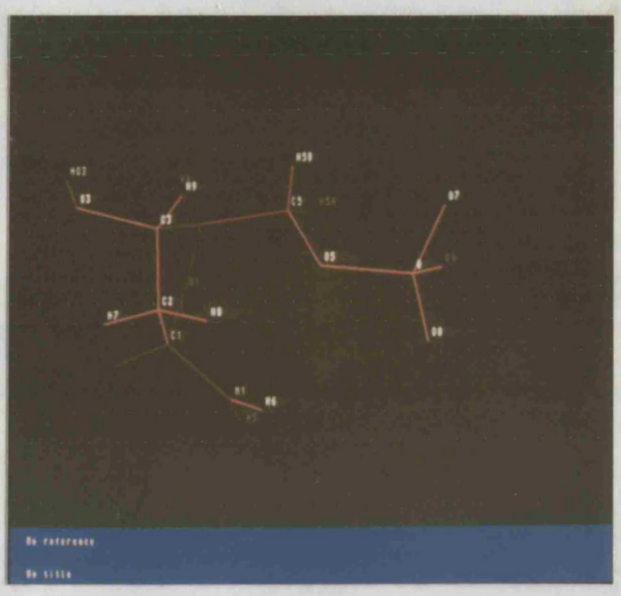
ALL LABELLING TURNED ON

BEST VIEW OPTION

FIGURE 7.5 TOP LEVEL MENU OPTION FUNCTIONS



ALL HYDROGENS TURNED OFF



DEPTH CUEING FUNCTION

FIGURE 7.5 continued TOP LEVEL MENU OPTION FUNCTIONS

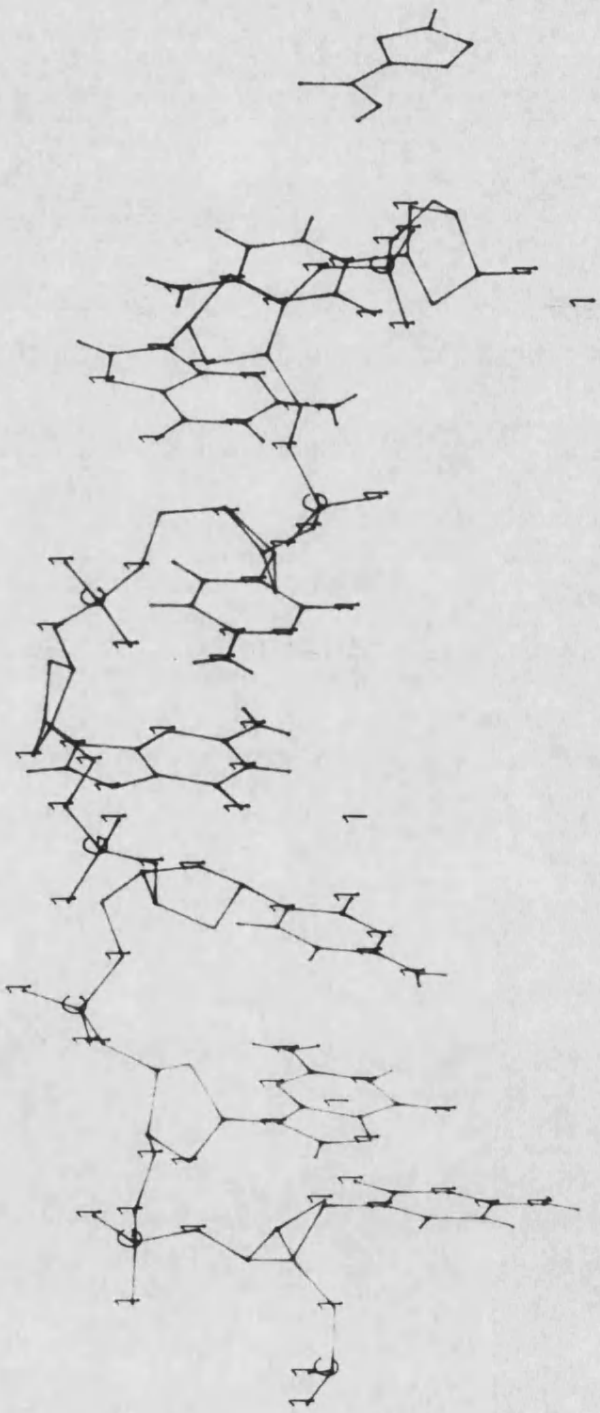
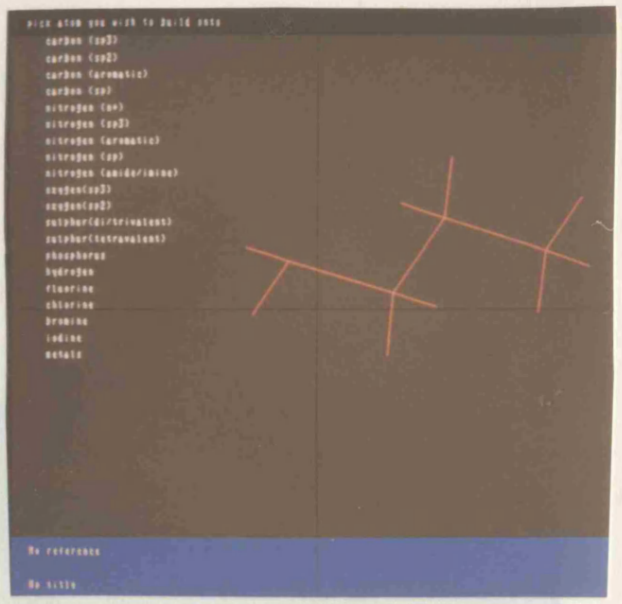
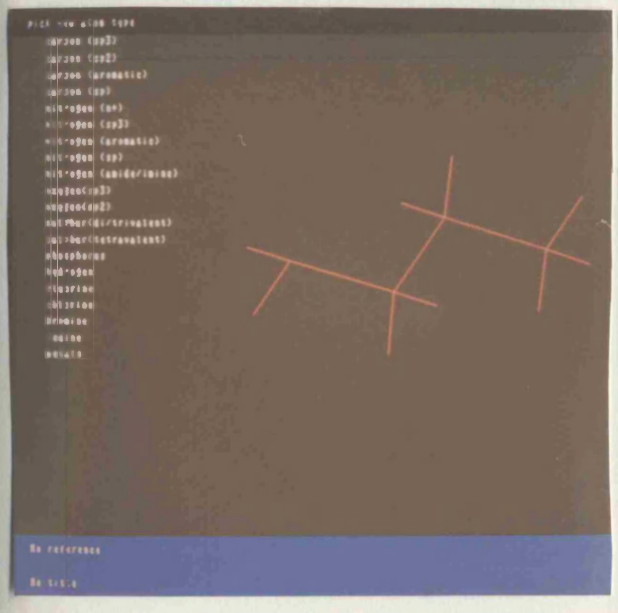


FIGURE 7.6 HARDCOPY OUTPUT FROM THE GLXMOD SYSTEM

H ATOM REMOVED
FROM C4 ATOM



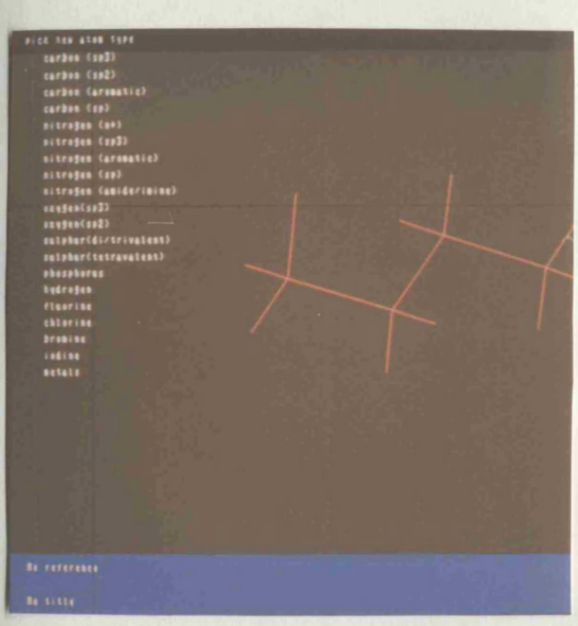
SELECTION OF C4
ATOM TO BULID ONTO



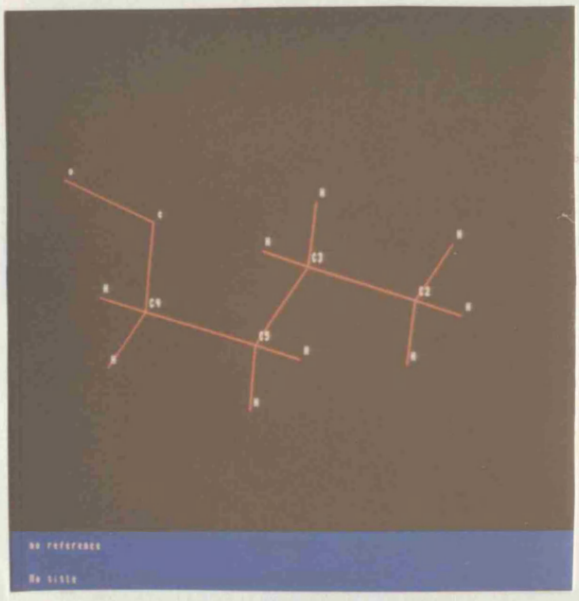
SELECTION OF ATOM
TYPE TO BE ADDED

PLACING OF NEW C ATOM
& SELECTION OF NEXT
ATOM TO BUILD ONTO

FIGURE 7.7 A BUILD SEQUENCE USING BUTANE

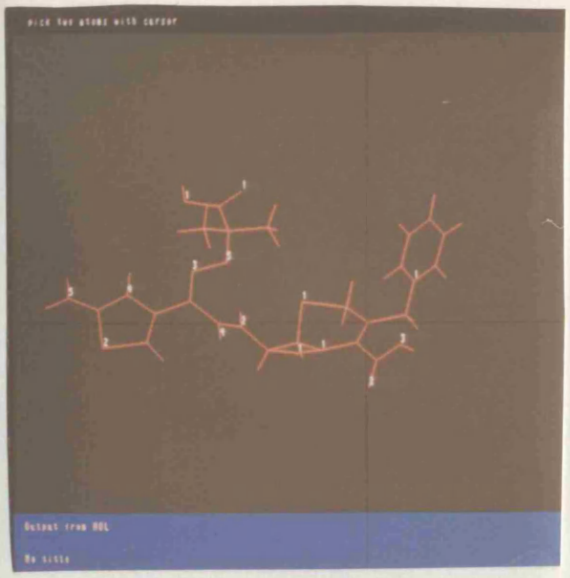


SELECTION OF ATOM
TYPE TO BE ADDED

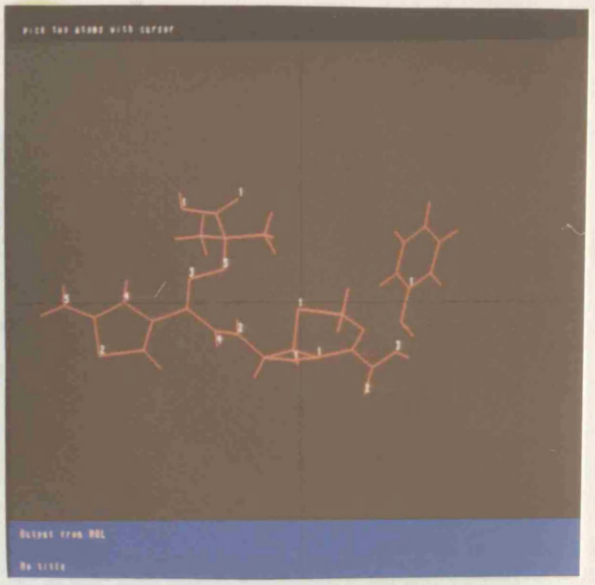


FINAL MOLECULE SHOWING
ADDITION OF C & O ATOMS

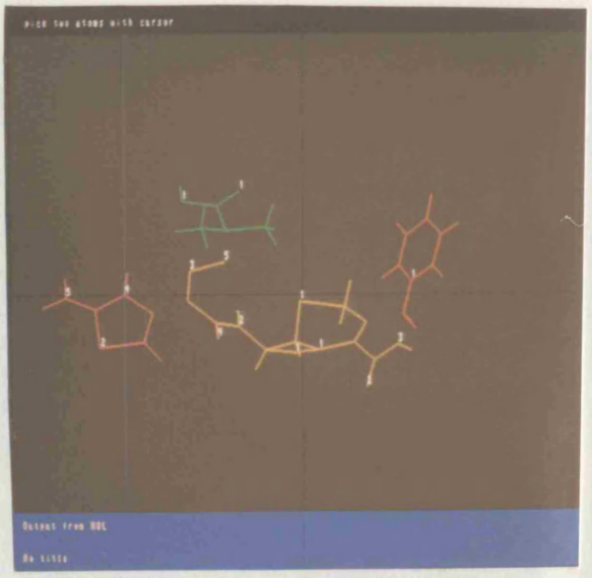
FIGURE 7.7 continued A BUILD SEQUENCE USING BUTANE



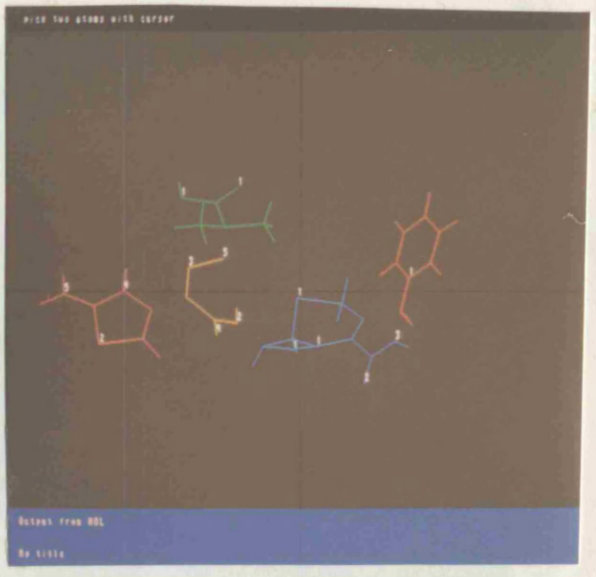
COMPLETE MOLECULE



2 FRAGMENTS



4 FRAGMENTS



5 FRAGMENTS

FIGURE 7.8 USER SELECTABLE FRAGMENTATION

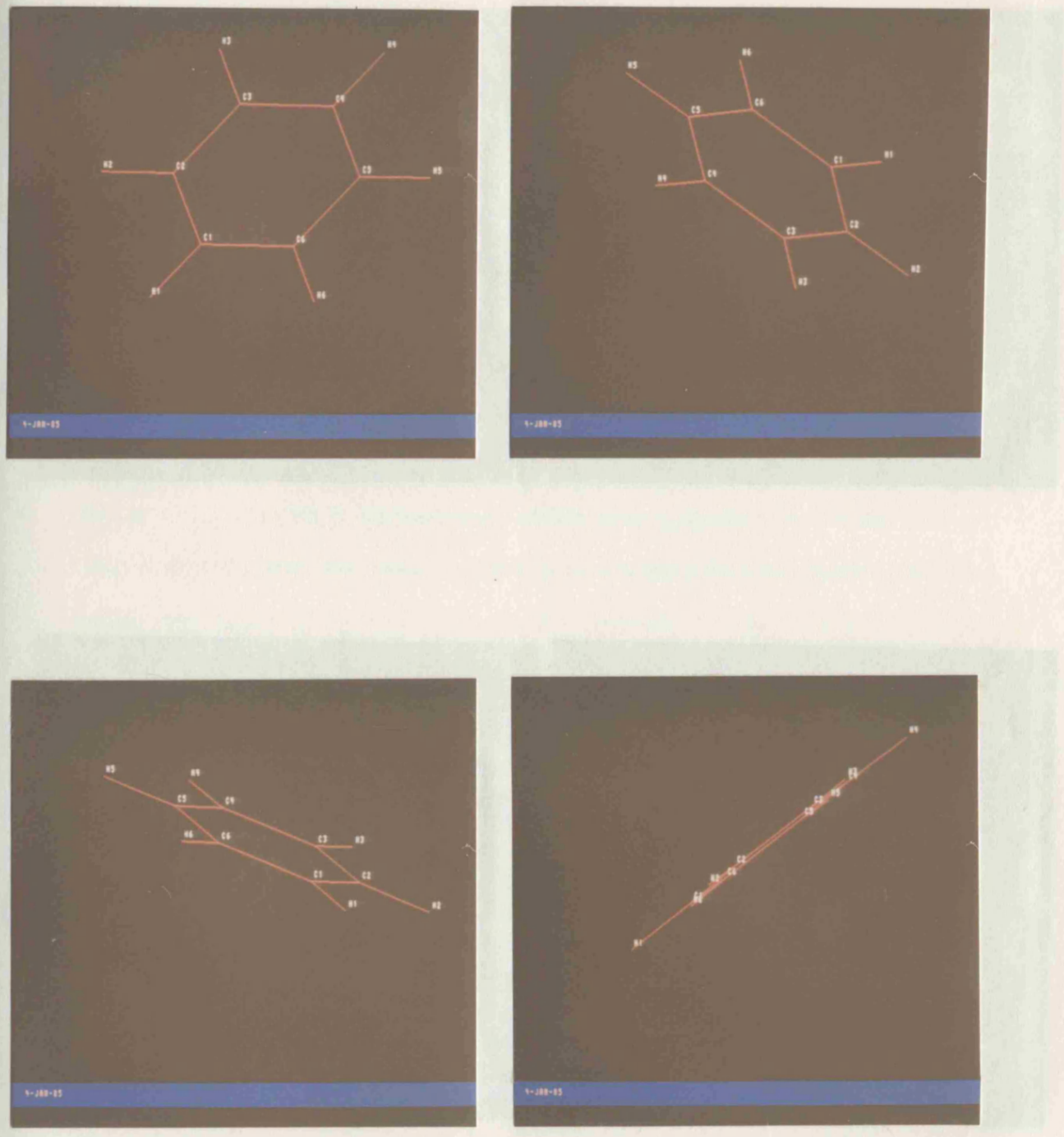


FIGURE 7.9 ROTATION SEQUENCE OF A BENZENE MOLECULE

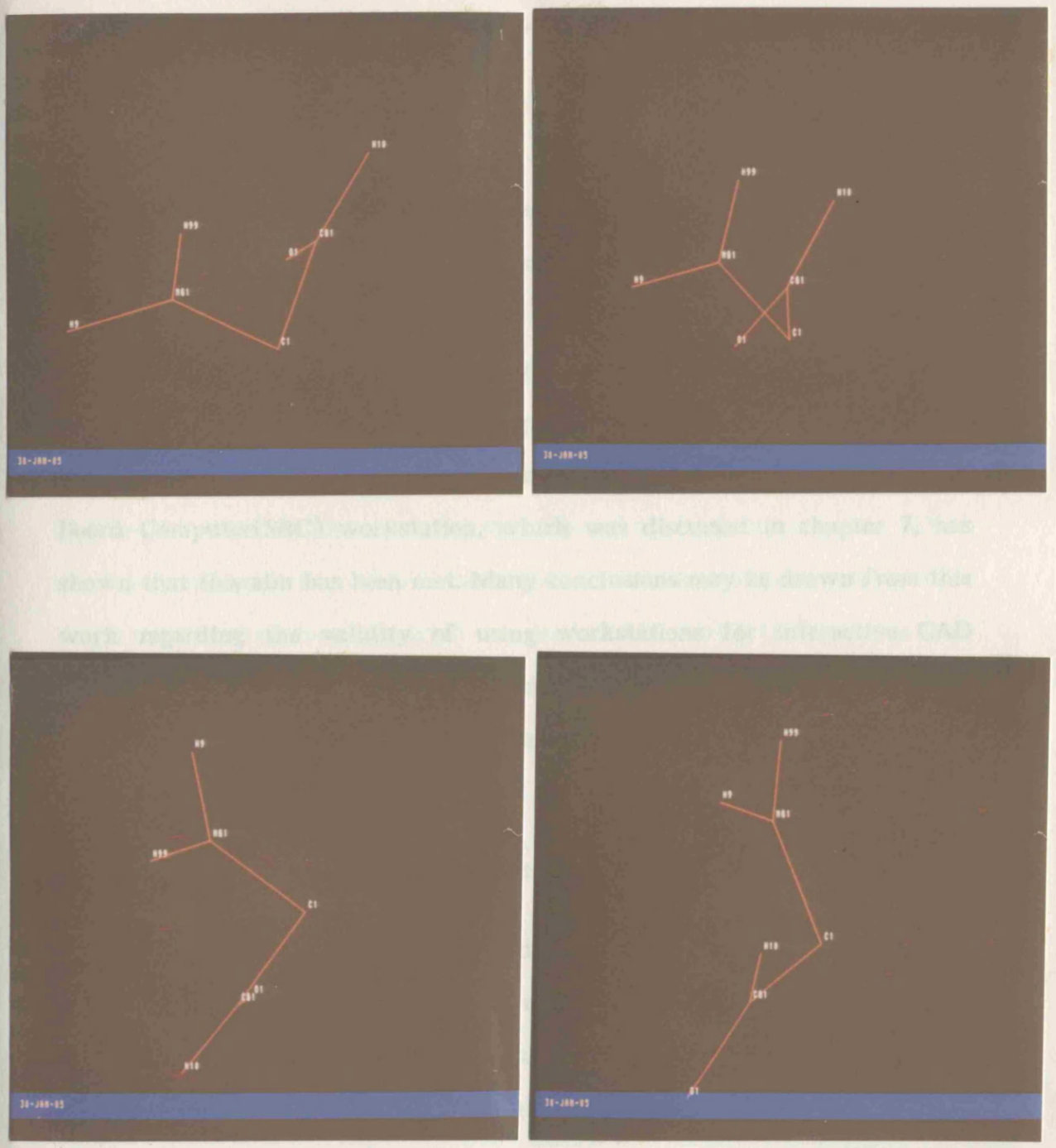


FIGURE 7.10 ROTATION SEQUENCE OF A GLYCENE MOLECULE

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

8.1 Conclusions.

The aim of this study has been to produce a microprocessor based workstation that supports a standard operating system with comprehensive facilities to allow the execution of non-trivial CAD application programs. These programs are written in a standard high level programming language for which a GKS interface library exists to address the graphical input and output features of the machine. The porting of a molecular modelling package onto the Single Board Computer(SBC) workstation, which was discussed in chapter 7, has shown that this aim has been met. Many conclusions may be drawn from this work regarding the validity of using workstations for interactive CAD applications and how appropriate the use of the Graphical Kernel System(GKS) is in addressing the needs of graphics programmers while ensuring the portability of their programs.

8.1.1 Standard Interfaces Promote Portability.

Using FORTRAN77 and GKS proved to be a successful combination for portability. The hardware peculiarities regarding character and file based input and output are shielded by the semantics of FORTRAN, while the graphical hardware peculiarities are shielded by the GKS virtual device model. Whereas the workstation operators perception of character and file based I/O is very similar on different hardware configurations, his interaction as achieved through graphical input and output is very dependent on the features of the

display device and its input controls. GKS proved very capable in ensuring that these dependencies did not affect the Man Machine Interface(MMI) as designed by the CAD application programmer. Hence, standard interfaces are essential and help induce program and programmer portability. The functionality of GKS also proved sufficient in supporting the specification, generation and manipulation of images for the molecular modelling program described in chapter 7.

8.1.2 Interactive CAD on Microprocessor based Workstations.

This work has also proven that single user microprocessor workstations can now provide the per-user processing power of mainframe machines at a fraction of the cost. Their memory addressing range and the size of disc storage also enable them support large operating systems and applications programs. These implications suggest that a move away from mainframe based systems towards single user workstations connected to a central file serving facility by a local area network should be made. The workstation designer must therefore be concerned with all the aspects of a system that were previously distributed amongst many specialist suppliers such as computer manufacturers, graphics terminal manufactures and local area networkng manufacturers.

The use of distributed processing proved very successful in improving interaction with the SBC workstation. The simulation of GKS input device classes using the BBC computer improved the speed of response of the workstation dramatically. Using the HD63484 advanced CRT controller offloaded most of the GKS primitive simulation that would otherwise have been done by the workstation microprocessor. Where standards such as GKS are employed, the use of distributed microprocessors can help bring the functionality of a physical device up to that required by its specification in the

virtual domain without loading the main applications processor. Hence, where possible, microprocessors should be used to map virtual devices onto their underlying physical devices.

8.2 Recommendations for Future Work.

The SBC workstation is functionally complete regarding its hardware system components which therefore leads in the short term to recommendations regarding the enhancement of its software system components. For example, the RAL GKS library could be extended to support level 2c functionality, and the input device emulator running on the BBC computer could be extended to support simultaniously active input devices and to manage the GKS input event queues. Other short term activities could involve the porting of further CAD application programs to the SBC workstation. Additionally, an application could be developed on the SBC workstation, and this ported to a different GKS environment.

The SBC workstation does have have a non-standard interface regarding its networking. An Ethernet[68] interface, rather than the Multi-Link system currently used would represent more the de-facto standard for local area networking. This could be achieved by fitting an Ethernet card to the SBC workstation or by using a Multi-Link to Ethernet transition box.

8.2.1 Using Powerful Microprocessors to Improve Performance.

A more long term view regarding the hardware components of a workstation must involve the use of the latest VLSI devices. This must of course be constrained to the use of low cost, mass produced devices to ensure that the single user workstation philosophy remains competitive with that of the multi-user mainframe approach. Considering firstly general purpose

microprocessors, the Motorola MC68020 is now widely available, as is the National Semiconductors NS32000[69] and the Intel 80386[70] device, all of which have full 32 bit architectures. Any new workstation design should use one of these devices, or even the next generation of microprocessors such as the Motorola MC68030[71]. An order of magnitude improvement in performance should easily be attainable over the MC68010 based SBC workstation for the following reasons. Firstly, the enhanced instruction set leads to an easier and more efficient translation from high level languages such as FORTRAN or C into machine code. Secondly, the provision to support floating point co-processors such as the MC68881 removes the need for software simlation of floating point calculations which is particularly important in CAD and GKS applications where most data objects are modelled using floating point values. And finally, the large data paths increase bus bandwidth and permit faster data transfer around the system.

8.2.2 Increased Performance through Distributed Architectures.

Parallel processor architectures constructed from devices such as the Transputer[72] may one day become cheap enough to be readily integrated into low cost workstations. However, the best solution today is to use distributed processing based on conventional microprocessors and their support devices such as the HD63484 advanced CRT controller. This solution, as applied by the SBC workstation could be enhanced in any subsequent re-designs. A more tightly coupled multi-microprocessor system which communicated via a common backplane would be constructed from the following distributed components. Firstly, the main system processor would run the CAD application program, and communicate with a general purpose input/output processor and a graphics input/output processor. The general I/O processor would handle the

disc drives, networking hardware and serial communications ports. Communication between the main processor and the I/O processor should be designed at a high level to minimise the number of transactions and hence minimise backplane usage. The graphics I/O processor would also communicate with the general purpose I/O processor, mainly to obtain access to permanent disc based storage. Input devices such as mice and joysticks should be directly handled by the graphics I/O processor since the interpretation of their values via echoing and transformations is semantically related to the graphics processing task. This will again help reduce backplane traffic and improve throughput. Communication between the graphics I/O processor and the main processor should be modelled on the GKS interface, or some higher level interface such as 3D-GKS[73] or PHIGS[74] to maintain standardization. The graphics I/O processor should use a CRT controller to manage the bit-mapped frame store and drawing operations. The CRT controller should support as closely as possible the drawing of primitives as specified by GKS to completely offload primitive simulation tasks from the graphics I/O processor. This may become a reality, since some work has already been done to produce a single VLSI device that has GKS functionality[75].

The final recommendations concerning the hardware aspects of a new workstation are connected with the quality and resolution of its display. By using ECL integrated circuits, a resolution of the order 1024x1024 pixels could be achieved using non-interlaced scanning. A low persistence monitor could therefore be used, which would be essential to avoid smearing on a system capable of fast picture updates.

8.2.3 A PHIGS Interface for CAD Applications.

GKS has been an I.S.O. standard since 1983, and since that time further effort

has been expended to provide 3-D extensions to GKS and also to produce a standard that is based on GKS, but which gives the application programmer a much richer and more powerful method of modelling and manipulating graphics objects. This new system is called PHIGS(Programers Hierarchical Graphics System). In the past, only very specialised and powerful graphics terminals have been capable of supporting PHIGS, but with the distributed system described above, it may become possible to support PHIGS on a low cost single user workstation.

The recommendations for future work listed above are no doubt currently keeping many workstation manufacturers very busy. When work commenced on the SBC workstation, standardisation in small workstation systems was not a strong commercial issue, and hence the production of an open workstation that could run UNIX and support GKS using microprocessor based hardware and which could also support large CAD applications was an important topic for investigation. Today however, the low cost workstation approach is very popular and companies understand the advantages to be gained from standardising their system interfaces. This has led to massive investment in research by these companies, thus taking the design of workstations from the realms of the University environment.

REFERENCES

1) J. Mick, J. Brick: Bit-Slice Microprocessor Design.
McGraw-Hill, New York, 1980.

2) Motorola: MC68020 32-Bit Microprocessor Users Manual.
Prentice-Hall, inc., Englewood Cliffs, N.J. 07632, 1984.

3) Motorola: MC68881 Floating Point Co-processor Users Manual.
Prentice-Hall, inc., Englewood Cliffs, N.J. 07632, 1984.

4) I. E. Sutherland: Sketchpad A Man-Machine Graphical Communication
System. Garland Publishing Inc., New York London, 1980.

5) C. Machover: A Brief Personal History of Computer Graphics.
Computer, Vol 11, No. 11, Nov 1978, pp. 38-45.

6) C. Machover: Background and Source Information About Computer Graphics.
Computers and Graphics, Vol 2, No. 2, 1977, pp. 119-122.

7) Independent Broadcast Authority: Specification of Television Standards
for 625-Line System I Transmissions. IBA Technical Review, London, 1977.

8) W. Myers: Staking Out the Graphics Display Pipeline.
IEEE Computer Graphics and Applications 7 (1984), pp. 60-65.

9) J.D. Foley, A. van Dam: Fundamentals of Interactive Computer Graphics.
Addison-Wesley, Reading, MA. 1981.

10) I. E. Sutherland, G. W. Hodgman: Reentrant Polygon Clipping.

Communications of the ACM 1 (1974) 17 pp. 32-42.

11) J. E. Bresenham: Algorithm for Computer Control of a Digital Plotter. IBM Systems Journal, Vol 4, No 1, 1965, pp 25-30.

12) D. Cohen: On Linear Difference Curves. Dept. Eng. Appl Maths, Harvard University. 1969.

13) W. M. Newman, R. F. Sproull: Principles of Interactive Computer Graphics. McGraw-Hill Inc., New York, 1979.

14) Hitachi: HD6845 CRT Controller User's Manual. Hitachi Electronic Components Ltd, Harrow Middlesex, 1976.

15) Thomson-Efcis: EF9365/66 Graphics Display Processor Data Sheet. Thomson-Efcis, Velizy, France, 1979.

16) NEC Electronics: Product Description of the uPD7220 Graphics Display Controller. NEC Electronics, 1982.

17) Hitachi: HD63484 ACRTC User's Manual. Hitachi Electronic Components Ltd, Harrow Middlesex, 1984.

18) RUNIT Computer Centre: GPGS-F User's guide. University of Trontheim, Norway, 1975.

19) The GINO-F Support Team: GINO-F User's Manual. Issue 2, CAD Centre, Cambridge, 1975.

20) SIGGRAPH ACM: Graphics Standards Planning Committee Status Report. Computer Graphics ACM, Vol 11, No 3, 1977 and Vol 13, No 3, 1979.

21) F. R. A. Hopgood, D. A. Duce, J. R. Gallop, D. C. Sutcliffe:
Introduction to the Graphical Kernel System (GKS).
Academic Press Inc., London, 1983.

21a) P. Bono, J. Encarnacao, F. R. A. Hopgood, P. J. W. ten Hagen: GKS -
The First Graphics Standard. IEEE Computer Graphics 7 (1982) pp. 9-23.

21b) Graphical Kernel System (GKS), Functional Description.
Draft International Standard ISO/DIS 7942, (ISO TC97/SC5/WG2/N163),
ANSI, 1983.

22) B. W. Kernighan, D. M. Ritchie: The C Programming Language.
Prentice-Hall Inc, New Jersey, 1978.

23) Motorola: MC68000 16-Bit Microprocessor User's Manual.
Second Edition, Motorola Inc, 1980.

24) D. G. Tanner: Real-Time Simulation of Power Systems.
Ph.D. Thesis, University of Bath, 1982.

25) I. G. R. J. Roker: A Real-Time Operating System.
Ph.D. Thesis, University of Bath, 1986.

26) Motorola: MC68450 Direct Memory Access Controller Data Sheet.
Motorola Ltd, East Kilbride, Glasgow, 1986.

27) Motorola: MC68451 Memory Manager Data Sheet.
Motorola Ltd, East Kilbride, Glasgow, 1986.

28) Motorola: MC68230 Parallel Interface and Timer Data Sheet.
Motorola Ltd, East Kilbride, Glasgow, 1986.

29) Motorola: MC68681 Dual Universal Asynchronous Receiver Transmitter Data Sheet. Motorola LTD, East Kilbride, Glasgow, 1986.

30) L. A. Dale: Real-Time Simulation of Multi-Machine Power Systems. Ph.D. Thesis, University of Bath, 1986.

31) Micrology pbt Inc.: VMEbus Specification Manual, Revision C.1. Printex Publishing Inc., 1985

32) M. Richards, A. R. Aylward, P. Bond, R.D. Evans, B. J. Knight: TRIPOS - A Portable Operating System for Mini-computers. Software-Practice and Experience, Vol 9, (1979), pp. 513-526.

33) M. Richards, Colin Whitby-Strevens: BCPL the language and its compiler. Cambridge University Press, Cambridge, 1982.

34) K. Moody, M. Richards: A Coroutine Mechanism for BCPL. Software-Practice and Experience, Vol 10, (1980), pp. 765-771

35) T. King: Tripos Technical guide. University of Bath, School of Electrical Engineering, 1982.

36) K. Thomson: UNIX Implementation. Bell Laboratories, Murray Hill, New Jersey, 1979.

36a) D. M. Ritchie: The UNIX I/O System. Bell Laboratories, Murray Hill, New Jersey, 1979.

36b) M. R. M. Dunsmuir, G. J. Davies: Programming the Unix System. Macmillan Publishers Ltd., London Basingstoke, 1985.

37) S. R. Bourne: The UNIX System. Addison-Wesley, 1982.

38) S. R. Chandler: A High Resolution Graphics System for the MC68000 Computer System, Final year project, University of Bath, 1983.

39) Electronic Industries Association: EIA Standard RS-434. Washington DC., 1969.

40) Motorola: MC68121 Intelligent Peripheral Controller User's Manual. Motorola Ltd, East Kilbride, Glasgow, 1979.

41) Motorola: MC6801 Microprocessor User's Manual. Motorola Ltd, East Kilbride, Glasgow, 1978.

42) Motorola: M68000 Family Resident Structured Assembler Reference Manual. Motorola Semiconductor Products Inc. Phoenix, Arizona. 1984

43) Motorola: RT120 Real-Time Monitor for 68120. Motorola Inc., 1980.

44) Fairchild: FAST Fairchild Advanced Schottky TTL. Fairchild Camera and Instrument Corporation, California, 1980.

45) Texas Instruments: The TTL Data Book, Parts 1 and 2. Texas Instruments, 1973, and 1977.

46) Mitsubishi: C-6920 High Resolution Colour Monitor. Mitsubishi Electric Corporation, Tokyo, 1984.

47) Euroquartz: Product Reference Manual. Euroquartz Ltd, Crewekerne, Somerset, 1984.

48) Texas Instruments: TM4164EK8 65,536 by 8-Bit Dynamic Ram Module data sheet, Texas Instruments, 1984.

49) John Birkner: PAL Programmable Array Logic Handbook.
Monolithic Memories GMBH, Munich, 1983.

50) D. J. Doornink, J. C. Dalrymple: The Architectural Evolution of a
High-Performance Graphics Terminal.
IEEE Computer Graphics and Applications 4 (1984), pp. 47-54.

51) Data-Type: Data-Type Terminal User's Reference Manual. 1978.

52) Gould: Colourwriter Operators Manual.
Gould Bryans Instruments Ltd, 1983.

53) Simon Chandler: The TGKS Language Binding.
School of Electrical Engineering, University of Bath, 1984.

54) N. Rowe: Investigation into the use of Computer Graphics in Planning Facial Surgery.
Final Year Project, University of Bath, 1985.

55) P. A. Keevil: Design and Development of a Graphics Display System for use
with Computer Models of Large Electrical Machines.
Final Year Project, University of Bath, 1985.

56) N. Hatzigianakis: Three Dimensional and Menu Driven Interactive Computer Graphics.
M.Sc. University of Bath, 1985.

57) Electronic Industries Association: EIA Standard RS-432.
Washington DC., 1969.

58) Signetix: 6522 Versatile Interface Adaptor User's Guide., 1979.

59) Signetix 6502: Microprocessor User's Guide., 1979.

60) J. Coll: The BBC Microcomputer User Guide.
British Broadcasting Corporation, London. 1982.

61) Rutherford Appleton Laboratory: Workstation Driver Manual RAL/ICL
GKS Implementation, Rutherford Appleton Laboratory, Chilton, Didcot
Oxon, 1984.

62) F. M. Lillehagen: CAD/CAM Work Stations for Man-Model Communication.
IEEE Computer Graphics and Applications 1 (1981) 3, pp. 17-27.

63) R. A. Watts: Introducing Interactive Computing.
NCC Publications, Manchester, 1984.

64) Catalogue of CPK Precision Molecular Models.
Ealing Corp., S. Natick, Mass., 1981.

65) C. Leventhal: Molecular Model Building by Computer.
Scientific American (1966) 214 pp. 42-53.

66) Digital Equipment Corporation: Introduction to VAX/VMS.
Digital Equipment Corporation. 1984.

67) Chemical Designs Ltd: The Chemgraph Chemical Modelling System.
Chemical Designs Ltd, Oxford, Oxon, 1982.

68) Digital Equipment, Intel, Xerox: The Ethernet. A Local Area Network,
Data Link Layer and Physical Layer Specification. Version 1.0, 1980

69) National Semiconductors: NS32032 High Performance Microprocessors.
National Semiconductors, Santa Clara, California, 1983.

70) Intel: iAPX80386 32-Bit Microprocessor Users's Manual.

71) Motorola: MC68030 Second Generation 32-Bit Enhanced Microprocessor.
Motorola Ltd, East Kilbride, Glasgow, 1986.

72) INMOS: IMS T212 Transputer, product description.
INMOS Ltd, Bristol, 1985.

73) G. Enderle, K. Kansy, G. Pfaff: Computer Graphics Programming.
Springer-Verlag, Berlin Heidlberg New York Tokyo, 1984.

74) S. S. Abi-Ezzi, A. J. Bunshaft: An Implementer's View of PHIGS.
IEEE Computer Graphics and Applications 2 (1986), pp. 12-23.

75) M. E. Mehl, S. J. Noll: A VLSI Support for GKS.
IEEE Computer Graphics and Applications 8 (1984), pp. 52-55.

APPENDIX A

MONITOR TIMING CALCULATIONS FOR THE IGP SYSTEM

Typical monitor timing specifications for medium resolution systems of approximately 780*780 pixel resolution with non-interlaced scanning.

Assuming a pixel clock frequency, $P_f = 30$ MHz

and a pixel clock period, $P_t = 3.34 * 10^{-8}$

Typical vertical frequency, $V_f = 60$ Hz

giving a vertical period, $V_t = 0.0166$ msec

Typical horizontal frequency, $h_f = 30$ KHz

giving a horizontal period, $h_t = 33\ \mu s$

Typical horizontal retrace period, $h_r < 7\ \mu s$

Typical vertical retrace period, $V_r < 650\ \mu s$

Typical video bandwidth = 25 MHz

The maximum number of visible lines per video frame is defined as,

$$\frac{V_t - V_r}{h_t} = \frac{16.7 - 0.65}{0.033} = 513\ lines$$

The maximum number of pixels per active horizontal scan line can then be calculated as,

$$\frac{h_t - h_r}{P_t} = \frac{33\times10^{-6}-7\times10^{-6}}{3.4\times10^{-8}} = 764\ pixels$$

APPENDIX B

SHARED MEMORY DATA FORMAT FOR THE IGP PROGRAMMING INTERFACE

This appendix lists the functions performed by the IGP system described in chapter 4. The offset parameter refers to the offset from the base of the common shared memory segment.

Note: p1xH,p1xL represent the upper and lower bytes of the X screen coordinate.
p1yH,p1yL represent the upper and lower bytes of the Y screen coordinate.
Xmax and Ymax are the maximum display X and Y coordinates respectively.

Function #1 - Draw a series of connected vectors as defined by their endpoints.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	0	1	6
Line type i.d.	0-4	1	7
Linewidth scaling	n/a	1	8
Vector colour	0-255	1	9
Vector bufer size	8-80	1	10
	p1xH,p1xL	2	11
Vector endpoint data	p1yH,p1yL	2	13
	:		
	:		
(Data Buffer)	pnxH,pnxL	2	11+4n
	pnyH,pnyL	2	13+4n

Function #2 - Draw a single marker at a specified position.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	1	1	6
X position	p1xH,p1xL	2	7
Y position	p1yH,p1yH	2	9
Marker Type	0-4	1	11
Marker Scaling	n/a	1	12
Marker Colour	0-255	1	13

Function #3 - Plot a Single Character at a Specified position.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	2	1	6
X position	p1xH,p1xL	2	7
Y position	p1yH,p1yL	2	9
Character i.d.	(ascii character set)	1	11
Character size	0-15	1	12
Drawing Direction	0-7	1	13
Character Colour	0-255	1	14

Function #4 - Program the Colour Pallette.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	3	1	6

Colour number	0-255	1	7
Red Magnitude	0-15	1	8
Green Magnitude	0-15	1	9
Blue Magnitude	0-15	1	10

Function #5 - Program the Display Zoom Factor.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	4	1	6
Zoom Factor	0-15	1	7

Function #6 - Program the Display Start Address(Scroll)

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d	5	1	6
X Start Position	p1xH,p1xL	2	7
Y Start Position	p1yH,p1yL	2	9

Function #7 - Position the Cross-Hair Cursor

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d	6	1	6
X Cursor Position	p1xH,p1xL	2	7
Y Cursor Position	p1yH,p1yL	2	9
Horizontal size	0-Xmax	2	11
Vertical size	0-Ymax	2	13

Function #8 - Program the Character Set Bit Patterns.

Parameter Name	Parameter Range	Parameter Size(Bytes)	Offset
Function i.d.	7	1	6
Character i.d.	(ascii character set)	1	7
Bit pattern #1	0-255	1	8
Bit pattern #2	0-255	1	9
Bit pattern #3	0-255	1	10
Bit pattern #4	0-255	1	11
Bit pattern #5	0-255	1	12
Bit pattern #6	0-255	1	13
Bit pattern #7	0-255	1	14
Bit pattern #8	0-255	1	15

APPENDIX C

DISPLAY RESOLUTIONS FOR A 60MHz PIXEL CLOCK FREQUENCY

Horizontal resolution, $h_r = 1280$

Vertical resolution, $V_r = 1024$

With the following monitor timings,

Pixel clock frequency, $p_f = 60\text{MHz}$

Pixel period, $p_t = 16.67$ n/seconds

therefore the active horizontal video period will be,

$$h_r \times p_t = h_a = 1280 \times 16.67 \times 10^{-9} = 21.34\ \mu sec$$

The typical horizontal retrace period for raster scan monitors in this class is 6 μsec.

Hence total horizontal period, $h_t = 21.34\ \mu\text{sec} + 6\ \mu\text{sec} = 27.34\ \mu\text{sec}$

giving a horizontal frequency, $h_f = 1/27.34 \times 10^{-6} = 36.58$ KHz

The active vertical video period will be,

$$V_r \times h_t = V_a = 1024 \times 27.34 \times 10^{-6} = 0.028 \text{ seconds}$$

The typical vertical retrace is 0.7 μsec

Hence total vertical period, $V_t = 28 \times 10^{-3} + 0.7 \times 10^{-3}\ \mu\text{sec} = 28.7 \times 10^{-3}$ seconds

giving a vertical frequency, $V_f = 1/28.7\times10^{-3} = 34.8$ Hz.

To avoid flicker a vertical frequency in the range 50-60 Hz is required, hence interlaced scanning will be required to display, of the order of, 1280×1024 pixels with a 60MHz pixel clock.

APPENDIX D

TIMING SPECIFICATION FOR THE HD63484 GRAPHICS SYSTEM

All timings are calculated from the pixel clock frequency, which was chosen to be 55MHz.

Hence p_f = 55MHz giving a pixel period p_t = 18.2 n/sec.

System resolution is 1024 × 1024 pixels.

Timing calculations are based on Mitsubishi C-6920 [46] monitor timing specifications.

Active line period = $h_r \times p_t = h_a = 1024 \times 18.2 \times 10^{-9} = 18.64\ \mu sec$

with a horizontal retrace period of 10 μsec

Total line period, $h_t = h_a + 10\mu$ sec = 28.64 μsec

Therefore line frequency, $h_f = 1/28.64 \times 10^6 = 34.9$ KHz

Active vertical field period using interlaced scanning =

$$h_t \times \frac{V_R}{2} = 28.64 \times 10^{-6} \times \frac{1024}{2} = 0.01467 seconds$$

with a vertical retrace period of 1 msec

Therefore field period, $f_t = 0.01467 + 1 \times 10^{-3} = 0.0157$ seconds

Therefore field frequency, $f_f = 1/0.0157 = 63.8$ Hz

The clock frequency to the HD63484 is given by,

$$\frac{P_f}{8} = \frac{55\times10^6}{8} = 6.875MHz$$

The memory cycle time for display cycles is given by,

$$\frac{2}{2\times CLK} = \frac{2}{6.875\times10^6} = 290n\,/sec$$

APPENDIX E

PALASM SOURCE LISTING FOR BLANK

INPUT SIGNALS		OUTPUT SIGNALS	
ASSIGNMENT	PIN No	ASSIGNMENT	PIN No
MCYC	1	/SD	13
/HSYNC	2	FB1	14
/VSYNC	3	FB2	15
/DRAW	4	FB3	16
/AS	5	FB4	17
/DISP1	6	/BL	18
/VSRLD	7		
/BLANK	8		
I1	9		
I2	11		
I3	12		
I4	14		

EQUATIONS

/SD = /HSYNC./VSYNC.MCYC./DRAW./I2 +

/HYSNC./VSYNC.MCYC.BLANK./DISP1

FB1 = /AS./I1

FB2 = /I2./HSYNC./VSYNC./DRAW./VSRLD

FB3 = /I3./HSYNC

FB4 = /I4./DISP1

/BL = /I3./DISP1

APPENDIX F

THE TGKS DI/DD INTERFACE SPECIFICATION

All parameters passed are assumed to represent integer values. Error checking must be made within the DD code to ensure that the values passed do not exeed the capabilities of the physical devices.

If appropriate, the global variables ip1x, ip2x, ip1y and ip2y contain the coordinates of the primitive to be dawn.

Function 1 - dev.initdevice()

Input Parameters - None.

Output Parameters - None.

Action - To initialize the workstation description table entries for the other device dependent functions. To load a TRIPOS driver if the workstation requires it.

Function 2 - dev.uninitdevice()

Input Parameters - None.

Output Parameters - None.

Action - To deallocate the TRIPOS driver if the device is no longer required.

Function 3 dev.clear()

Input Parameters - None.

Output Parameters - None.

Action - Clear the workstation display surface.

Function 4 - dev.drawline(type,scale,colour,flush)

Input parameters - type, the required line type.
scale, the required linewidth scale factor.
colour, the required colour number.
flush, a boolean variable indicating the end of this
primitive.

Output parameters - None.

Action - To draw the requested line using the attributes given by exploiting the features of the device to give as close a simulation as possible.

Function 5 - dev.drawmarker(type,scale,colour)

Input parameters - type, the required marker type.
scale, the required marker scale factor.
colour, the required colour number.

Output parameters - None.

Action - To draw the requested marker using the attributes given by exploiting the features of the device to give as close a simulation as possible.

Function 6 - dev.char(chrs,chht,chwd,colour,attvec)

Input parameters - chrs, character string to be plotted.
chht, required character height.

chwd, required character width.

colour, required colour number.

attvec, text up vector.

Output parameters - None.

Action - To plot the requested text string using the attributes given by exploiting the features of the device to give as close a simulation as possible.

Function 7 - dev.intensity(red,green,blue,ired)

Input parameters - red, normalised red intensity value.

green, normalised green intensity value.

blue, normalised blue intensity value.

Output parameters - ired, three element vector containing the device specific representation of the cooresponding red, green and blue intensities respectively.

Action - To return the device specific representation of the normalised intensity values of red, green and blue.

Function 8 - dev.colour(number,red,green,blue)

Input parameters - number, the colour number.

red, magnitude of the red intensity.

green, magnitude of the green intensity.

blue, magnitude of the blue intensity.

Output parameters - None.

Action - To select the intensity magnitudes cooresponding to the specified colour number.

APPENDIX G

COMMUNICATION PROTOCOL FOR THE INPUT/TERMINAL EMULATOR SYSTEM

A character based protocol is used to control input and output from the BBC computer. An input operation is initialised by sending a hex 1C control character to the terminal. The interaction is completed, either by the operator issuing a successful input or a break action.

Input class for interaction is indicated by the following characters.

"L" - Select LOCATOR for input.

"C" - Select CHOICE for input.

"V" - Select VALUATOR for input.

Physical device selection is achieved using the following characters.

"K" - Select keyboard for input.

"B" - Select absolute bitstick for input.

"M" - Select mouse for input.

"V" - Select velocity bitstick for input.

Protocol for device selection.

Initialise an input action - 1C

Select input class for interaction - ["L"],["C"],["V"]

Select device type - "D"

Choose physical device - ["K"],["B"],["M"],["V"]

Protocol for window size selection.

Initialise an input action - 1C

Select input class for window - ["L"]

Initiate window specifiation - ["W"]

Top 5 bits of X max window with bit 6 set

Bottom 5 bits of X max window with bit 6 set

Top 5 bits of Y max window with bit 6 set

Bottom 5 bits of Y max window with bit 6 set

Top 5 bits of X min window with bit 6 set

Bottom 5 bits of X min window with bit 6 set

Top 5 bits of Y min window with bit 6 set

Bottom 5 bits of Y min window with bit 6 set

Protocol for initial value selection.

Initialise an input action - 1C

Select input class for initial position - ["L"],["C"],["V"]

Initiate initial position specification - "I"

Following characters for LOCATOR specification.

Top 5 bits of X position with bit 6 set

Bottom 5 bits of X position with bit 6 set

Top 5 bits of Y position with bit 6 set

Bottom 5 bits of Y position with bit 6 set

Following characters for CHOICE specification.

Menu number with bit 6 set

Character string for menu option #1 terminated by CR

-

-

Character string for menu option #n terminated by LF

Following characters for VALUATOR specification.

Character string for tag #1 terminated by CR

-

-

Character string for tag #n terminated by LF

Protocol to start input emulation

Initialise an input action - 1C

Select input class for this emulation - ["L"],["C"],["V"]

Execute emulation - "E"

APPENDIX H

UNIX DRIVER INTERFACE FOR THE HD63484 GRAPHICS SYSTEM

The UNIX driver for the HD63484 graphics system assumes communication with a character device. Once this device has been opened, the following functions become available for graphical output.

Function 0 - To clear the display surface.

Parameter	Value
Function identifier	0

Function 1 - To draw a series of connected vectors using the specified attributes.

Parameter	Value
Function identifier	1
Linetype	1 - 5
Linewidth	not used
Left clipping boundary	0 - 1023
Bottom clipping boundary	0 - 1023
Right clipping boundary	0 - 1023
Top clipping rectangle	0 - 1023
Number of coordinate pairs	2 - n
X1	0 - 1023
Y1	0 - 1023
-	-
-	-

Xn	0 - 1023
Yn	0 - 1023

Function 2 - To draw a series of markers using the specified attributes.

Parameter	Value Range
Function identifier	2
Markertype	1 - 5
Marker scale factor	1 - 16
Left clipping boundary	0 - 1023
Bottom clipping boundary	0 - 1023
Right clipping boundary	0 - 1023
Top clipping rectangle	0 - 1023
Number of coordinate pairs	1 - n
X1	0 - 1023
Y1	0 - 1023
-	-
-	-
Xn	0 - 1023
Yn	0 - 1023

Function 3 - To program the bit mapped font table.

Parameter	Value range
Function identifier	3
Position in font table	0 - 255
Number of characters.	1 - 256
Bit map (1,1)	0 - 255

Bit map (1,2)	0 - 255
Bit map (1,3)	0 - 255
Bit map (1,4)	0 - 255
Bit map (1,5)	0 - 255
Bit map (1,6)	0 - 255
Bit map (1,7)	0 - 255
Bit map (1,8)	0 - 255
-	-
-	-
Bit map (n,1)	0 - 255
Bit map (n,2)	0 - 255
Bit map (n,3)	0 - 255
Bit map (n,4)	0 - 255
Bit map (n,5)	0 - 255
Bit map (n,6)	0 - 255
Bit map (n,7)	0 - 255
Bit map (n,8)	0 - 255

Function 4 - To plot a string of characters using the specified attributes.

Parameter	Value range
Function identifier	4
X string position	0 - 1023
Y string position	0 - 1023
X incremental value	0 - 1023
Y incremental value	0 - 1023
Character width	0 - 1023

Character height	0 - 1023
Colour	0 - 255
Left clipping boundary	0 - 1023
Bottom clipping boundary	0 - 1023
Right clipping boundary	0 - 1023
Top clipping rectangle	0 - 1023
Number of characters in string	1 - n
Character 1	0 - 255
-	-
-	-
Character n	0 - 255

Function 5 - To fill the specified polygon using the cooresponding attributes.

Parameter	Value range
Function identifier	5
Fill style	not used
Fill colour	0 - 255
Left clipping boundary	0 - 1023
Bottom clipping boundary	0 - 1023
Right clipping boundary	0 - 1023
Top clipping rectangle	0 - 1023
Number of vertices	3 - n
X1	0 - 1023
Y1	0 - 1023
-	-
-	-

Xn	0 - 1023
Yn	0 - 1023

Function 6 - To draw the specified cell array.

Parameter	Value range
Function identifier	6
X dimension	0 - 1023
Y dimension	0 - 1023
Left clipping boundary	0 - 1023
Bottom clipping boundary	0 - 1023
Right clipping boundary	0 - 1023
Top clipping rectangle	0 - 1023
X cell dimension	0 - 1023
Y cell dimension	0 - 1023
X position	0 - 1023
Y position	0 - 1023
cell 0 colour number	0 - 255
-	-
-	-
cell n colour number	0 - 255

Function 7 - To initialise the HD63484device.

Parameter	Value range
Function identifier	7

Function 8 - To uninitialise the HD63484 device.

Parameter	Value range
Function identifier	8

Function 9 - To directly program the registers of the HD63484 device.

Parameter	Value range
Function identifier	9
Register identifier	0 - 7
Data to be written to register	0 - 65535